INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Master's Thesis

Performance Predictions for Large-Scale Data Applications

Maximilian Höb

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Master's Thesis

Performance Predictions for Large-Scale Data Applications

Maximilian Höb

Supervision:	Prof. Dr. Dieter Kranzlmüller
Advisors:	Dr. Andre Luckow Dr. Nils gentschen Felde
Date:	November 27th, 2017

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 27. November 2017

. . . . . .
(Unterschrift des Kandidaten)

Abstract

The cloud computing community is constantly developing new applications, job submitting frameworks and data processing paradigms. In all areas, new approaches are recognized, which should enable each user to profit from this fast development. To support users in their decision, which of the huge amount of different cloud configurations is the optimal for a current application, this thesis introduces a methodology to predict the runtime behavior of any large-scale data application within a cloud. Therefore, significant features of these configurations are varied during runtime measurements of the underlying application, which resulting values will be mathematically investigated within a function analysis. The main statistical method will be a Multiple Linear Regression, which will value the relation between the selected features. Based on them, a runtime prediction and a configuration selection is possible. The developed methodology will be instantiated on two use cases, the clustering algorithm K-Means and Wordcount, using Apache Flink as a job processing framework.

Contents

1 Introduction **1**
1.1 Structure of the Thesis . . . 2

2 Related Work **5**
2.1 Prediction Approaches . . . 5
2.1.1 Performance Prediction Framework . . . 5
2.1.2 Predicting the best Cloud Configuration . . . 5
2.1.3 Kernel Based Predictions . . . 6
2.1.4 Predictive Scheduling . . . 6
2.2 Evaluation of the Related Work . . . 7

3 Methodology to predict Runtime Behavior **9**
3.1 Problem Space and Data arrangement . . . 9
3.2 Function Analysis . . . 12

4 Methodology Instantiation **13**
4.1 Execution environment . . . 13
4.1.1 Apache Flink . . . 13
4.1.2 LRZ Compute Cloud . . . 15
4.2 Problem Space and Measurements . . . 17
4.2.1 Use Case K-Means . . . 18
4.2.2 Measurements . . . 19
4.2.3 Dataset definition . . . 23
4.2.4 Adjusted R^2 value . . . 24
4.2.5 Dimensions' transformations . . . 24
4.2.6 Determining Statistical Significance of Dimensions . . . 27
4.2.7 Problem Space Refinement . . . 31
4.3 Function Analysis . . . 32
4.3.1 Regression . . . 32
4.4 Multiple Linear Regression Model . . . 34
4.4.1 Outlier and Leverage point treatment . . . 34
4.4.2 Linear Regression Coefficients . . . 36
4.4.3 Linear Regression Representation . . . 39
4.5 Validation of the Regression Model . . . 43
4.5.1 Normal Distribution of Measurements . . . 44
4.5.2 Dimension's Correlation and Independence . . . 44
4.5.3 Confidence and Prediction Interval . . . 46
4.5.4 Evaluation of the Instantiation . . . 49

5 Runtime Prediction and Regression Model Evaluation **51**
5.1 Prediction . . . 51

5.2 Configuration Prediction . . . 52
5.3 Validation of a Regression Model . . . 53
5.4 Use case K-Means . . . 54
5.4.1 Application of the Methodology . . . 54
5.4.2 Runtime Predictions . . . 54
5.5 Use case Wordcount . . . 57
5.5.1 Application of the Methodology . . . 58
5.5.2 Runtime Predictions . . . 61
5.6 Accuracy of Measurements and Predictions . . . 64

6 Evaluation and Conclusion **69**
6.1 Methodology Evaluation . . . 69
6.2 Outlook and Future Work . . . 70

A appendix **71**
A.1 Measured Runtime Values . . . 71
A.1.1 K-Means . . . 71
A.1.2 Wordcount . . . 77
A.2 Apache Flink Configuration . . . 82

List of Figures **83**

Bibliography **85**

1 Introduction

Recent trends in almost all areas of big data processing have to deal with an increasing amount of data and applications. Many scientific areas like meteorology, sociology or physics have their traditional models to compute their results, but newer applications, with new programming and executing paradigms on constantly increasing data can be observed.

As the computation powers increased worldwide in the last years, the possibilities of more computations but also of more complex computations grew with at last the same speed up. Additionally, the more complex the models became, the more data had to be processed. This data certainly also had to be stored somewhere. For this reason each computational environment included a storage for big data.

However, besides classical computer centers with professional maintenance of hardware and software, more commercial business centers found their way into cloud computations and the scientific areas, offering computation as a service. Everybody worldwide can rent their own super computer online, for a certain time, as long as their budget lasts. Today, this computational philosophy is known as cloud computing and one very profitable sector. It combines the need of computational power at a certain time with a cost shared maintenance of the underlying hardware. Instead of buying and maintaining their own hardware servers, nowadays scientists deploy and terminate virtual machines.

Frameworks like Apache Spark or Hadoop make it even simpler to start computations in such distributed systems. They offer job submission and controlling utilities, where any user needs only to submit his compiled code implementations and a corresponding configuration. In contrast to classical parallel programming models like the Message Passing Interface, any communication or synchronization within the application does not have to be implemented explicitly.

Classical parallel computing, where each programmer had to decide when and how a task is parallelized found no place in cloud environments. The new frameworks instantiated the parallelism automatically and even optimize the execution.

Today's scientific questions in this cloud computing sector more often belong not to the fields of hardware optimization, but rather software optimization. Processing frameworks have established their place in scientific research and a change in the data processing paradigm might slowly come up on the horizon. Thus analysis of data often becomes more important than the data itself.

The need to actually save any data at all is more often refused. Therefore, so-called data-streaming is also becoming increasingly essential in these computations. If data is only processed and not stored, it does not have to be sent to the cloud and made available there, but could instead be easily caught from any stream, available online or artificially created. It could even minimize the need for big data storages within the cloud environments and hence be one concern less a user has to think of when planning his computation.

However, this challenge can be transferred to a more general problem, applicable to data-streaming, but also to any other data based computing, namely the selection of an optimal configuration for any computation. Such criteria can follow several objectives. For many

users, the overall costs play an essential role. Some users may be concerned by the necessity of any running resources and may like to optimize their computation to reduce these resources in order to follow a green IT paradigm, but most users are concerned not if and how the computation is completed, but when. Time is usually the major factor and not always only for commercial users.

Time plays an essential role, for instance in an urgent computing situation. Knowing the temporal behavior of the executed application can not only reduce the runtime by extending the resources in the essential configurations, but also enables the optimization of the gradation of any computation. For example, every year hurricanes can be observed in several regions on the planet. All meteorological institutions contribute to predicting the direction and strength of such a storm. If it comes close to inhabited areas, it is essential and probably life-saving to have predictions as soon as possible, but also as accurate as possible, mostly depending on the granularity of the input data.

To fulfill this objective, it is absolutely necessary the know everything about the prediction model. For instance, on which configuration of how many virtual machines is the computation finished after one hour, and what is the maximum input to certainly achieve this runtime. Knowing these answers can improve each prediction, because simply combining more and more computational power will not at all necessarily accelerate any computation. Applications react very different and therefore need to be analyzed.

Therefore this thesis will present a methodology to characterize any data-intensive application within a cloud environment to describe and finally predict its runtime behavior. This methodology will enable a runtime prediction dependent on specified configuration values. Each configuration consists of values of significant parameters, for instance the number of virtual machines or the applications' input sizes. In addition, it will also be possible to identify an optimal configuration for a certain defined subset of the configuration, for example for a specified target runtime.

Based on runtime value measurements taken for relevant combinations of the defined problems pace, the main computation will rely on a Multiple Linear Regression. Together with other statistical methods like different information criteria or a Outlier detection, this function analysis will determine the statistically significant features whose relation builds the regression model. Its estimated solution will be the basis for any runtime prediction.

Although data-streaming applications and frameworks are not new to the scientific research, no known approaches exists which focuses a generic prediction model for data-streaming applications and frameworks. The existing models are either not applicable on other applications and especially on their input types, or on other job processing frameworks than the used ones.

This methodology introduced in this thesis can be generalized to multiple algorithms and applications, also including data-streaming. The instantiation of the methodology using K-Means and Wordcount as two representative examples of common data analytic algorithms will be demonstrated in this thesis.

1.1 Structure of the Thesis

Chapter 2 will present four other prediction approaches as the related work. It will be shown, that all these other approaches have been limited to special set ups and not include any possibility to adapt generic data-streaming.

The methodology will be outlined in chapter 3 providing an overview of all major steps and the underlying problem space. The main part of the function analysis depends on a statistical approach, namely a Multiple Linear Regression. The methodology will be instantiated in chapter 4 on the application K-Means. Thereby, each part will be presented with its statistical and mathematical explanations and exemplarily applied to the use case.

The basis of any runtime prediction with the estimated results of the regression model will be introduced in Chapter 5 and applied to the use cases K-Means and Wordcount. Therewith the gained instantiated regression models will be evaluated and it will be shown, that they can be validated. Finally Chapter 6 evaluates the complete methodology and presents future work related to this thesis.

2 Related Work

This chapter will evaluate published work on the same objective, namely to predict runtime of data-intensive applications or an optimal cloud configuration. In section 2.1 four other approaches will be presented and afterwards evaluated in section 2.2.

2.1 Prediction Approaches

2.1.1 Performance Prediction Framework

In [VYF+16] the authors around Venkataraman proposed a statical approach to predict the runtime of long running machine learning applications. The main idea is to measure the runtime of the applications with previously defined representative parts of the input files. The prediction result is taken to decide how many virtual machines are needed to complete a job run within a given objective. The prediction model can follow several user-defined cost functions to evaluate the best configuration.

Within their paper the authors implemented their framework within the Amazon EC2 cloud using Apache Spark 1.2 as a job submission tool. In contrast to the framework used in this thesis, Apache Spark supports at the moment of writing only batch data processing. The user workloads and applications are mostly machine learning algorithms of Apache Spark's scalable machine learning library, but also queries from GenBase, a complex analytics genomics benchmark, or a speech recognition pipeline. For all applications an input was selected on which each job was started. After 5% to 10% of the complete input file was processed, the runtime was measured and used as a data point for a non-negative least squares solver to learn values of their prediction model.

The model was build on four non-linear features, namely a cost term representing the serial computation runtime, the parallel computation time for algorithms scaling linearly with data, a term to model communication patterns and lastly a linear term to capture overheads of scaling by adding more machines to the execution system. The thereby developed model is able to predict the runtime for these specific applications within an average prediction error of under 20%. The authors define this as sufficient to choose an appropriate number or type of instances for the job execution while limiting the time spent on gathering the training data.

All predictions and executions were limited to predefined VM configurations within the used cloud. The focus of the prediction is on the number of VMs with the same configuration.

2.1.2 Predicting the best Cloud Configuration

The presented approach from Alipourfard and others in [ALC+17] tries to determine the best cloud configuration for a given, recurrent big data analytics job.

Their prediction model *CherryPick* uses Bayesian Optimization based on a few samples to estimate a confidence interval of the costs and running time of each tested cloud con-

figuration. The buildt prediction model does not have the aim of choosing an optimal configuration, rather a near-optimal configuration to avoid much overhead.

Their approach is only applicable to repeating jobs, where the cost of searching a configuration can be amortized during several subsequent runs. It is evaluated on five big data applications on overall 66 different cloud configurations. The authors state that *CherryPick* can find a near-optimal configuration within 5% of the optimal at the median with 45-90% chance. Compared to the previously presented work of *Ernest*, this approach shall reduce the search time by 90%.

The five applications used are benchmarks based on Apache Spark and Hadoop. They cover database queries, machine learning workloads and a clustering algorithm.

2.1.3 Kernel Based Predictions

Escobar and others present in their paper [EB16] an approach originally designed for High Performance Computing (HPC) clusters, but in future possibly applicable to cloud environments. Their model predicts execution times of parallel scientific applications using empirical analyses of the application runtimes for small input sizes and the time spent on various phases of the execution. The estimation of the execution time is based on benchmark kernels, which are assigned to the different phases of the execution. A regression approach is used to predict the overall execution time. Following the paper this approach requires only a few short executions of the application ,each less than a minute, to produce accurate execution time predictions. The model included only three applications with a more complex scientific background, like a solver for linear systems or a model of a neutral particle transport.

The authors state that their prediction error ranges from 1% to 15%. However, as the presented approaches before, this also requires a set of representative small input sizes, proving a good characterization of the general job executions.

2.1.4 Predictive Scheduling

The paper [LTX16] presents an approach to predict runtime performances of data-streaming applications based on the framework Apache Storm. Additionally the authors Li and others introduce a scheduling algorithm to assign the application's tasks under the guidance of their prediction results.

The developed prediction model predicts the average tuple processing time of an application. Such a data-stream processing system, according to the authors, handles unbounded streams of data tuples, which last for a long time. Therefore, the time between when a tuple is collected from a data source and when its processing is completed is used as the performance metric, the so-called tuple processing time. For its prediction the model also requires the selected scheduling solution (assignment of threads to workers and their physical or virtual machines), according to the topology of the application graph.

The presented model is evaluated on three representative applications, Wordcount, LogStream (read data from log files) and continuous query on a database, all examples of the Apache Storm framework. The experimental results show the topology-aware prediction method offering an average accuracy of 84% and the predictive scheduling framework reducing the average processing time by 35% compared to Apache Storm's default scheduler.

2.2 Evaluation of the Related Work

Ernest and CherryPick designed performance models for a limited number of applications with distinct preconditions.

Although Ernest covers several different applications, it requires a specific representative input file for each of them. This would be especially difficult including data-streaming input, which can vary in an arbitrary way. However, also for many batch processing applications, as stated in [ALC+17], it would be very hard to extend Ernest, because for instance the presented database query benchmark includes almost 100 queries depending on completely different sets of database tables, and each query is not comparable to the others. Thus, it would be difficult to determine which query is representable.

In this case, Ernest would build several models to pick the best configuration, but could take 11 times the search time of CherryPick.

CherryPick, presented in [ALC+17], on the other hand, was executed on specific benchmarks included in the used Apache Spark framework. This is different from real world applications which usually have no optimal resource usage. CherryPick relies on representative workloads of the used applications to suggest a cloud configuration, for instance parsing daily log files. Therefore, it may be applicable on such a specified use case, but cannot be simply transferred to other applications.

The presented approach in [EB16] of selecting the optimal kernels for different phases of an application execution might also be transferred to a cloud environment as stated by the authors. However, the problem of the main idea of changing an actual kernel might be limited with the compatibility of any cloud job processing framework. If there are significant differences between such frameworks running on different kernels needs to evaluated in the future. This could eventually also be a contribution to the performance model of the thesis by adding different kernels to the feature set.

For data-streaming applications not much work was published at the time of writing. The introduced paper [LTX16] is one that faces data-streaming. It is strongly based on Apache Storm, because it alters the specific sequence of the application parts with its scheduler. Reducing the runtime compared to Apache Storm's normal scheduler allows the authors afterwards to predict the runtime of an execution. This approach is not extendable to any other framework then Apache Storm, because it relies completely on its architecture. In contrast to the model developed in this thesis, a generic future version is not intended.

It can be stated that in the moment of writing no other approach was published to develop a general model focusing on data-streaming applications and frameworks. In addition, all other prediction models make restrictions on their ability to be applicable to other applications and frameworks as the ones used in their work.

3 Methodology to predict Runtime Behavior

This chapter will present a methodology to characterize an application executed on a distributed cloud infrastructure regarding the runtime behavior. This methodology outlined in figure 3.1 will be divided into two main phases.

The first is an iterative process with the aim of determining the underlying problem space by gathering runtime measurements. Several different configurations of the underlying infrastructure and the application itself need to be taken into consideration. Therefore, section 3.1 defines the high dimensional problem space, which covers the possible set screws of these configurations. From this space configurable features will be selected to build the basis of the measurements. The obtained measurement data will be used to determine the statistically significant dimensions and the resulting problem space iteratively.

On this resulting problem space a function analysis will be applied as described in section 3.2. It will be mainly affected by a Multiple Linear Regression, which estimates the linear relationship between the dimensions and hence enables a runtime prediction.

3.1 Problem Space and Data arrangement

Before any mathematical analysis can be applied, the basis for this purpose must be clearly defined. This problem space includes arbitrary dimensions, generally everything that influences a cloud job execution. An excerpt of it is shown in figure 3.2. The most important dimension is the *runtime* itself, whose behavior this methodology is supposed to predict. These response and measurement dimensions need to be included in every problem space this methodology should be applied on.

The underlying cloud infrastructure is the most influencing and hence limiting respectively enhancing factor. Its dimensions include, for example, the overall *cloud computing platform*, the *spatial allocation* of the nodes and their *interconnect*, which is a huge influence for all network-bound applications. Other applications might be CPU-, memory- or I/O-bound and would therefore profit from the *capability of these cloud nodes*, especially their computing power or the attached hard drive disks.

These infrastructure dimensions are very important, but are usually not alterable by any user, except by the cloud administration. Therefore they are not included in the determination. Adjustable are all configurations of the virtual machines, which include the *operation system*, the amount of *CPUs*, *RAM* and *disk space*, as well as the overall *number of virtual machines*. Also the *processing framework*, which starts and controls the job execution within the cloud on the deployed virtual machines, sets boundaries to the overall runtime.

Regarding the job itself, the specification of the *application* with its *configuration* is also a major factor. Especially the size of the processed *input data* influences the runtime. This impact differs from one application to another. Hence this methodology cannot be applied to more than one application at a time.

This itemization of dimensions does not claim any totality and could be extended. However

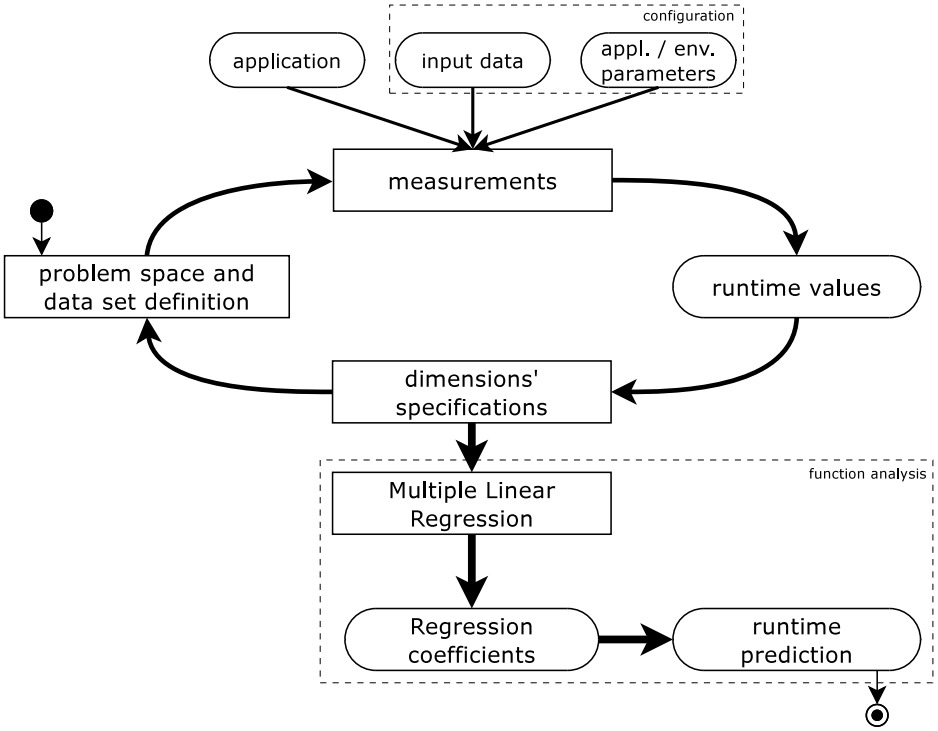


Figure 3.1: Graph of the methodology to gain the prediction parameter

these user configurable eight dimensions, dashed in figure 3.2, are supposed to build the superset of the problem space underlying this methodology.

Under the assumption that disk space is always large enough, the problem space focused on the configuration of the virtual machines and the application could be defined as shown in table 3.1.

runtime	measured / predicted
virtual machine	quantity
	allocated CPUs
	allocated RAM
	operation system
	processing framework
application	configuration
	input size

Table 3.1: Subset of the problem space focused on applications and virtual machines

This problem space is assumed to enhance a generic oriented methodology, because the deployment configuration of the virtual machines and the application itself is user defined in every cloud environment.

From these dimensions also only a subset can be chosen to limit the required measurements. These measurements need to be taken from all desired combinations of the them. Their number is also affected by the gradation of each dimension. While the operation system and the processing framework are usually limited and need to be combinable, the hardware

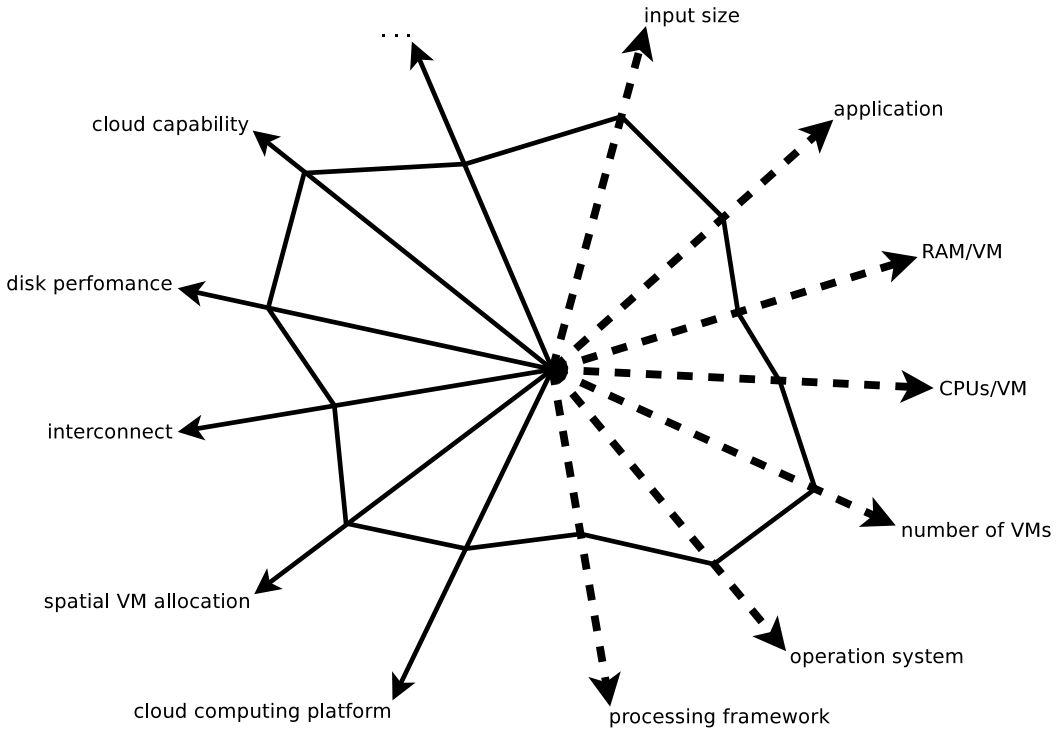


Figure 3.2: High-dimensional problem space for runtime analysis

configuration can be increased to the maximum capacity of the cloud. This methodology recommends to increase these dimension values by the power of 2 or 10. From a statistical point of view, linear increasing values are also possible and sufficient, only limited by the planned time spent on the measurements. The runtime of all these measurements needs to be collected together with the configuration of all other dimensions to build the data set.

Dimension's significance

In preparation of the function analysis, which will apply a Multiple Linear Regression on the response dimension *runtime*, all other included dimensions need to be investigated, if they are statistically significant. Therefore a simple Linear Regression is applied with the runtime being the response variable and the other dimensions each the prediction variable, following the regression model

$$runtime \sim \ f(dimension)$$

to specify the closest transformation $f(dimension)$ of the runtime values dependent on each dimension. From a predefined selection, these transformations will be determined by an algorithm.

With the *Akaike's Information Criterion*, detailed in the same section, the significance will be exposed. If one dimension shows no significant influence on the runtime, it should be disregarded to minimize the further measurements.

This iteration starts again with the reduced problem space, which can also be extended with a new dimension. This proceeding should be continued until the significance of all

dimensions is proved.

3.2 Function Analysis

For the function analysis an appropriate Multiple Linear Regression is chosen. Such a regression is a statistical approach to estimate the linear relationships between variables. In this case these variables are the defined dimensions of the problem space. The resulting regression function to be estimated can be describes as

$$\mathrm{Y} \sim \mathrm{X} \cdot \beta = \begin{bmatrix} 1 & X_1 & X_2 & \cdots & X_p \end{bmatrix} \cdot \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_\mathrm{p} \end{bmatrix} \tag{3.1}$$

where Y is the response variable (*runtime*), X contains the independent variables (the other dimensions) and β their regression coefficients. The selected regression will estimate these coefficients on the basis of the measured data, after a necessary review of Outlier and Leverage points.

Finally, these estimators of β will be used to predict new runtime values within a certain interval. Therefore new dimension values need to be applied to vector X in equation (3.1) returning a prediction runtime.

This methodology uses several mathematical definitions, which will all be deduced in the next chapter 4. In this chapter also the complete methodology will be instantiated on the use case K-Means. Correspondent predictions of this application and of a second use case Wordcount, as well as the evaluation of the methodology, will be presented in chapter 5.

4 Methodology Instantiation

This chapter will instantiate the defined methodology to predict runtime behavior on the example use case application K-Means, which is an iterative clustering algorithm, assigning given data points to optimized centroids.

The problem space for this thesis is defined in section 4.2, owing to the execution environment only including a subset of the introduced dimensions. It will be presented in section 4.1 introducing the used *LRZ Compute Cloud*, a shared infrastructure, and the used job processing framework *Apache Flink*.

To gather data for the function analysis runtimes of all combinations of the selected dimensions have to be measured within the described environment. To these results a function analysis needs to be applied to estimate the relationship between them. The mathematical idea is presented in section 4.3. As a result of this preliminary work, a Linear Regression is chosen to be the evaluation method, detailed in section 4.4.

This regression analysis consists of several analytic steps. The problem space dimensions and the gathered runtimes are combined with the basic dataset in section 4.2.3. Before a linear regression can be performed, the regression transformation needs to be determined for each dimension (section 4.2.5) and, if there is any statistical significance and contribution to the response value, the runtime (section 4.2.6). If not, the data set must be refined as described in section 4.2.7

Afterwards possible Outlier or Leverage points of the regression are identified in section 4.4.1, and it is assessed if they can be removed. Finally, the regression formula is verified and its coefficients can be calculated in section 4.4.2. The regression model is validated in section 4.5 at the end of this chapter.

4.1 Execution environment

For any runtime measurement, the basic cloud infrastructure is an important factor as stated. All computations and measurements in this thesis were executed on the *LRZ Compute Cloud*. It is detailed in section 4.1.2 with its hardware and operating system specifications. The still young processing framework *Apache Flink* was chosen to execute the applications. It will be introduced in section 4.1.1.

4.1.1 Apache Flink

Apache Flink is a young open-source system for processing streaming and batch data. It is a top-level project of the Apache Software Foundation since January 2015 and is a further development of the German Stratosphere project at TU Berlin[1].

This processing framework is built on the main idea that data processing applications like real-time analytics, batch data processing (static data) as well as machine learning algorithms

[1] `https://blogs.apache.org/foundation/entry/the_apache_software_foundation_announces69`

can be defined and executed as pipelined fault-tolerant dataflows. The following summary bases upon the findings of [CKE+15].

Data-stream processing and static batch data processing are traditionally considered as very different types of applications. The execution and implementation were done in separate systems. *Apache Storm* or *IBM Infosphere Streams* are two examples for dedicated streaming frameworks, while batch processing is used in execution engines for Hadoop like *Apache Spark* or *Apache Drill*.

The traditional batch data analysis was the leading data processing mentality, which was and is of course mainly used in most use cases. However, it is becoming more and more apparent, that nowadays a rising number of large-scale data processing use cases handle continuously produced data. These streams of data originated in web or application log files, technical sensor data, or in transaction log records of databases like any Twitter Stream.

Nevertheless, today's setups often ignore the continuous nature of data production and instead batch these records into static data sets, for example hourly or daily summaries, which are then processed in a time-ignoring fashion. Architectural patterns like the *lambda architecture* combine batch and stream processing systems, but suffer from high latency and complexity, as well as very variable inaccuracy, as time is not explicitly handled by the application code.

Apache Flink follows a new paradigm which integrates data-stream processing as the unifying model for continuous streams and batch processing, both in the programming model and in the execution engine. In addition to durable message queues, which allow quasi-arbitrary replay of data streams (for example in Apache Kafka or Amazon Kinesis), the Flink stream processing programs do not differentiate between processing the latest real-time events or terabytes of historically observed data. Instead, these different kinds of computations start at different points in the enduring stream, and maintain different states during their computations.

Yet Flink also acknowledges that there is, and will always be, a need for dedicated batch processing. Hence batch programs are treated as special cases of streaming programs, where the stream is finite, and the record's order and time is not relevant, within a specialized API for processing such static data sets.

As a result of the findings of [CKE+15], Flink presents itself as an adequate and efficient batch processor on top of a streaming runtime, including graph analysis and machine learning.

Apache Flink's architecture

As shown in figure 4.1 based on the results of [CKE+15], any Flink cluster consists of three types of processes: the controlling client, the job manager as the master node, and at least one task manager as a worker node. The client transforms the *Java* or *Scala* (the only two languages supported by Flink) program code to an optimized dataflow graph and submits it to the job manager, which coordinates and tracks the distributed execution of the dataflow. The actual data processing is done by the task managers, which execute the operators to produce streams and reports on their status as well as findings to the job manager. Flink has also integrated a checkpoint system to recover such a dataflow execution.

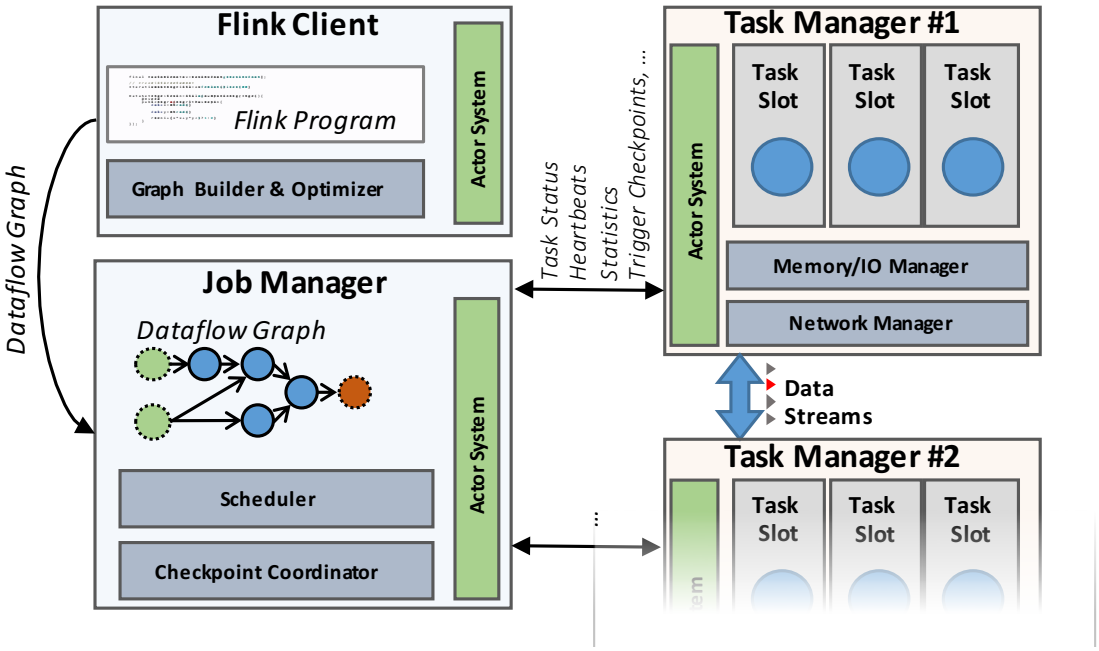


Figure 4.1: The Flink process model from [CKE+15]

4.1.2 LRZ Compute Cloud

The physical infrastructure that was used is the LRZ Compute Cloud[2]. The computation power[3] is defined by 95 nodes with several different configurations. The built-in CPUs (central processing units) are Intel Xeon E5540, X5650 or E5-2660v2 from which 1 to 8 are available on one node. The available RAM (random access memory) is 1 to 32 gigabytes on each computational node. The cloud usage is not exclusive, but is frequent shared, which will especially be important when evaluating the standard deviation of the measurements.

The cloud implementation is an OpenNebula 5.2.0 system. It offers the possibility to expose some functionalities by means of a RESTful interface, compatible with the Amazon Elastic Compute Cloud (EC2) API. With this interface the creation and termination of virtual machines, as well as information exportation from them like IP addresses or running status, are easily possible and can be also used in the framework explained in the next subsection.

All virtual machines are set up with Ubuntu 14.04.5 LTS (GNU/Linux 3.13.0-85-generic x86_64). For each configuration unique templates were created, which differ in the amount of CPUs per VM and RAM per VM and which can be instantiated with the RESTful interface. The number of virtual CPUs of a VM is always identical to the real number of CPUs per VM. Also for each application a unique image was created, which includes the appropriate input data sets, to minimize the traffic within the cloud and the startup time.

[2] https://www.lrz.de/services/compute/cloud_en/
[3] https://www.lrz.de/services/compute/overview/

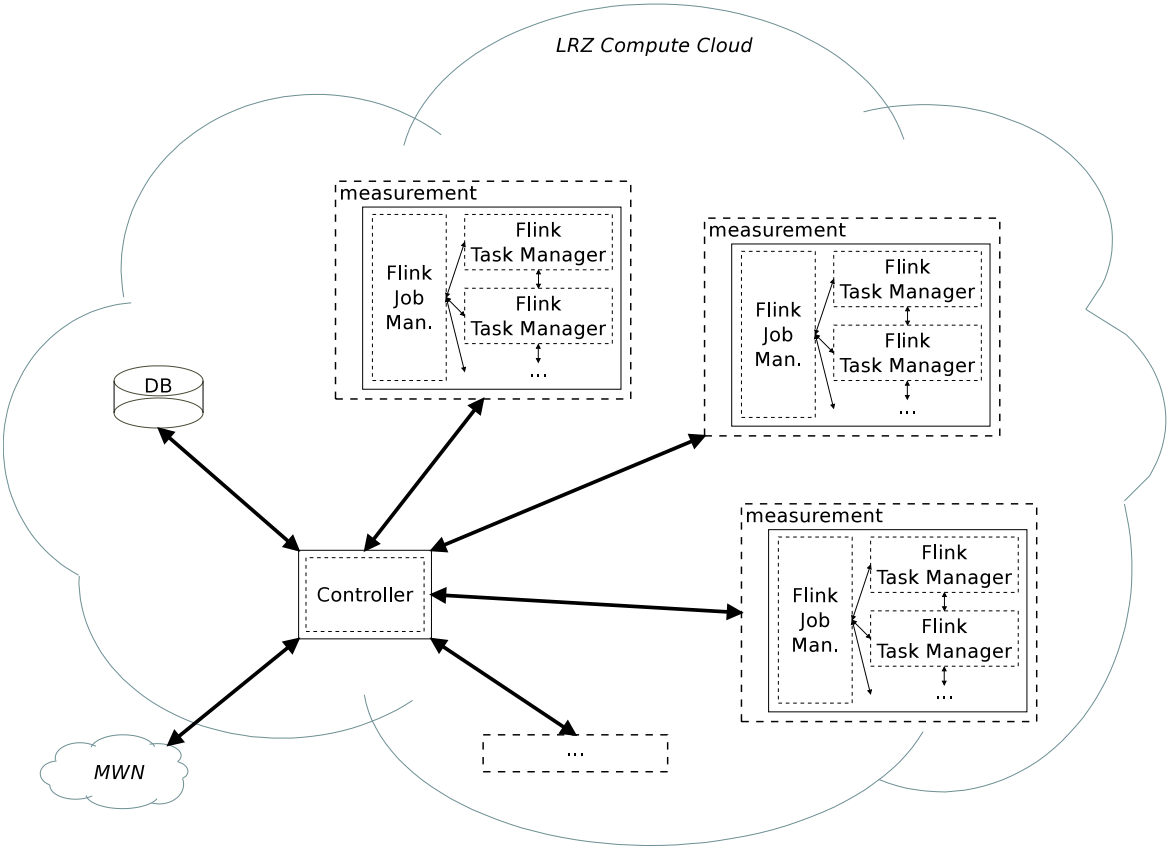


Figure 4.2: Experiment cloud setup infrastructure

Apache Flink was implemented on all images in version 1.3.2, the up-to-date build from August 2017. A shared file-system like Apache Hadoop was not used, instead all input files were local inputs on each worker and master of the Flink computation, which reduced the time delay of the execution to a minimum regarding the transfer of input files.

Automation framework

To run all needed measurements, control the virtual machines, gather the runtime results and save them into a database, and also to manage the datasets, a Python based framework was created, which was compatible to the RESTful interface of the OpenNebula cloud. It was so possible to run several measurements in parallel.

In figure 4.2 the complete setup can be seen. The cloud resources were only accessible within the Munich Scientific Network (MWN). Inside the cloud the controller, also a virtual machine, automatically sets up the Flink job and task manager virtual machines and starts the job execution. Afterwards the result is saved into a relational MySQL database.

For the huge amount of measurements several job executions with different configurations or applications were done simultaneously.

VMs	CPUs per VM	RAM per VM	input size
1	1	1	1 GB
2	2	2	10 GB
4	4	4	100 GB
8	8	8	
16		16	

Table 4.1: Problem space dimension values

4.2 Problem Space and Measurements

The introduced execution environment already specified some non-alterable problem space dimensions. As the processing framework *Apache Flink* was chosen, running on the operating system *Ubuntu 14.04.5 LTS*. With these two definitions both dimension are not alterable and hence removed from the problem space. Additionally, for this instantiation the application and its main configuration were fixed and are therefore also no longer included in the problem space. During the measurements only the input size was altered.

With the specifications of the *LRZ Compute Cloud*, also hardware boundaries were defined, because the cloud could not be physically changed. The virtual machines, for example, are deployed to nodes regarding the overall load and not concerning any spatial objectives between them. As stated, the cloud was also not exclusively used and so the hardware resources were limited to avoid blocking the whole cloud.

Therefore the remaining hardware dimensions could only affect the virtual machines (VMs) which were used as Flink Client, job manager and task manager (compare Figure 4.2). The three dimensions added to the problem space in this case were the *quantity of VMs*, the amount of *CPUs per VM* and the amount of *RAM per VM*.

The resulting, alterable four dimensions are listed in table 4.1, together with the values used in this instantiation that are expected to be significant. As can be seen all hardware dimensions are increased by powers of 2: 2^0, 2^1, and so on, and the input size of the application is increased by the powers of ten: 10^0, 10^1, and so on.

With the resulting *runtime* the five measurable dimensions are set. Yet for the later following Linear Regression, correlations of the prediction dimensions also need to be considered. One possibility would be to add all cross combinations $dim^r \times dim^r$, which would lead to a lot of unnecessary overhead in the mathematical analysis. However, with some logical examination, only such dimensions can correlate, which are affected by each other. In this case the input size is unaffected by all others, but the other dimensions could correlate due to the fact that they build one virtual machine together. For the Linear Regression, the problem space ($\mathrm{PS}_{\mathrm{LR}}$) consists of seven dimensions:

$$\mathrm{PS}_{\mathrm{LR}} = \{runtime, vms, cpus, ram, input, cpus \cdot vms, ram \cdot vms, cpus \cdot ram\}$$

This has no influence on the measurements, because the additional dimensions are only combinations of already stated measurements.

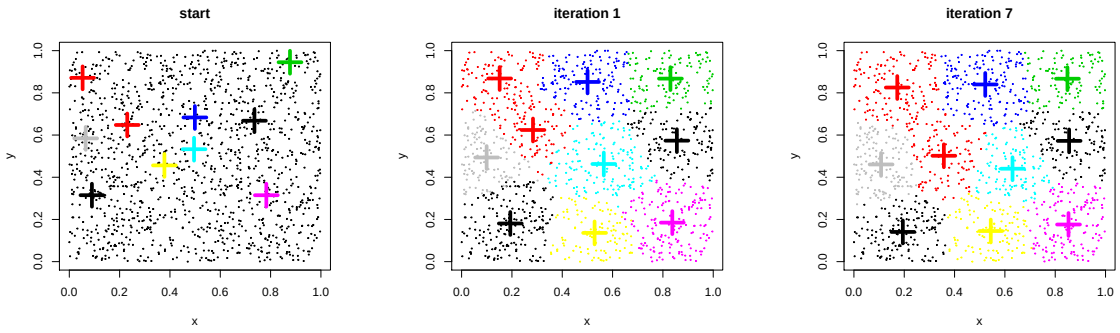


Figure 4.3: Example of K-Means iterations to find centroids

4.2.1 Use Case K-Means

Before the measurements are presented the use case application will be introduced. K-Means is an iterative clustering algorithm. It is defined in the description of the application in the Apache project documentation[4].

K-Means is given a two-dimensional set of data points with double values and a set of centroids, which define the first mean of the clusters. The aim is to find convergent cluster centroids and assign each data point to one of the clusters. In figure 4.3 the clustering is shown on an artificial data set. The first figure shows the initial centroids and the not clustered data points.

In each iteration, the algorithm computes the distance of each data point to each centroid. Hence each data point is assigned to the cluster which has the shortest distance from the centroid. After each iteration, the cluster centroids are shifted to the mean position of all data points which are assigned to it after the iteration.

With these new centroids, the next iteration is started. The algorithm terminates after a fixed number of iterations (default 10 if not initially specified) or if the cluster means do not significantly move in an iteration. Then the centroid converge to their final position.

The Flink implementation works on double values of data points and centroids, provided as plain text files, where each row contains two space separated double values. Additionally, the centroids file contains a leading index. The application computes an assignment of data points to centroids, where each data point is annotated with the index ID of the cluster it belongs to.

For the K-Means measurements in this use case the input sizes were defined as listed in table 4.2, according to the requirements of the methodology.

The file size of the points' text file is equivalent to the number of points used. To get a more tangible value, the number of data points was used instead of the file size, which is statistically allowed, like a scalar transformation. All points and centroids were random variables in the range from 0.000 to 100.000 each with three positions after the decimal point.

[4] `https://ci.apache.org/projects/flink/flink-docs-release-1.3/api/java/org/apache/flink/examples/java/clustering/KMeans.html`

points	centroids[5]	iterations[5]	file size	amount of measurements
5e+07	100	10	0.590 GB	184
20e+07	100	10	2.360 GB	52
50e+07	100	10	5.900 GB	151
200e+07	100	10	23.602 GB	40
500e+07	100	10	59.005 GB	23

Table 4.2: K-Means input sizes for measurements

4.2.2 Measurements

All measurements are performed in the described execution environment, which may not be changed during all measurements. It is also mandatory that the configuration of the Flink framework is not refined. The configuration used in this use case is listed in the appendix A.2.

If the complete problem space as described in table 4.1 and 4.2 would be covered, 500 measurements should be taken. Due to time constrains, not all combinations could be measured. Finally, 85 different configurations in 450 single measurements were taken, which are listed in the appendix A.1. To execute this still huge number of Flink runs, an automation like the framework described in section 4.1.2 is recommended, although it is of course not necessary. If this methodology is applied within a shared infrastructure, all measurements should to be taken several times to determine the deviation of the results. The smaller the deviation, the better the prediction possibility of the regression model.

Flink returns the overall runtime to the standard output of the client, which submitted the job to the job manager. It is embedded in a single line: *Job Runtime: 8764299ms*, where the runtime is given in milliseconds. It is also possible to fetch the runtime from the Flink RESTful API, which returns to completed detailed job information at the web client of the job manager with *job-manager:8081/joboverview/completed* referred to the API documentation[6].

In Figure 4.4 the complete series of measurements of the use case K-Means is shown. The pairwise dimension's plot was created with R[7]. At a first sight, the dimensional-wise plots show certain diffrent characteristics, which are elementary for the following Linear Regression. In the upper left plot for example, the number of VMs is plotted against the runtime. The shape of the theoretically placed curve might suggest a decreasing logarithmic behavior of the runtime values, while the input curve on the right bottom could suggest a linear increasing runtime value. These transformations will be determined in detail and used for evaluating the dimension's statistical significance later.

These four plots include all taken measurements, which makes it difficult to identify the runtime behavior directly. To give a better insight Figure 4.5 shows a Box-Whisker-Plot of the measured runtimes for only one input value. Here the number of VMs is plotted against the runtime and the input is fixed with 500 million points. The plot shows the mean runtime with their quartiles, indicting the variation of the measured values. The reduction of the runtime can be seen more precisely, but the plot also indicates a higher deviation of the

[5] fixed configuration values for all measurements

[6] `https://ci.apache.org/projects/flink/flink-docs-release-1.3/monitoring/rest_api.html`

[7] `https://www.r-project.org`

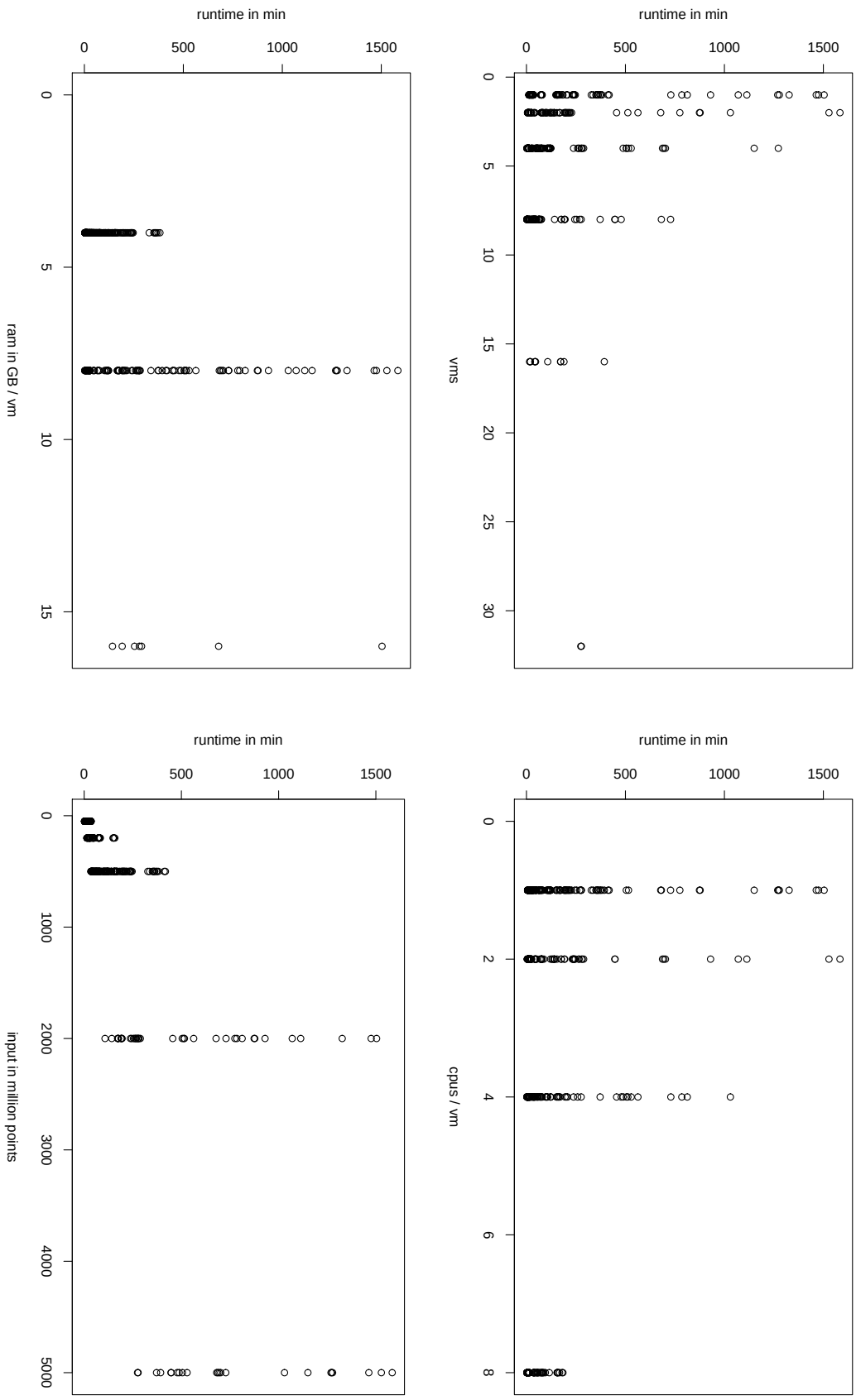


Figure 4.4: R-plot of the measurements in each dimension against runtime in minutes for the use case K-Means

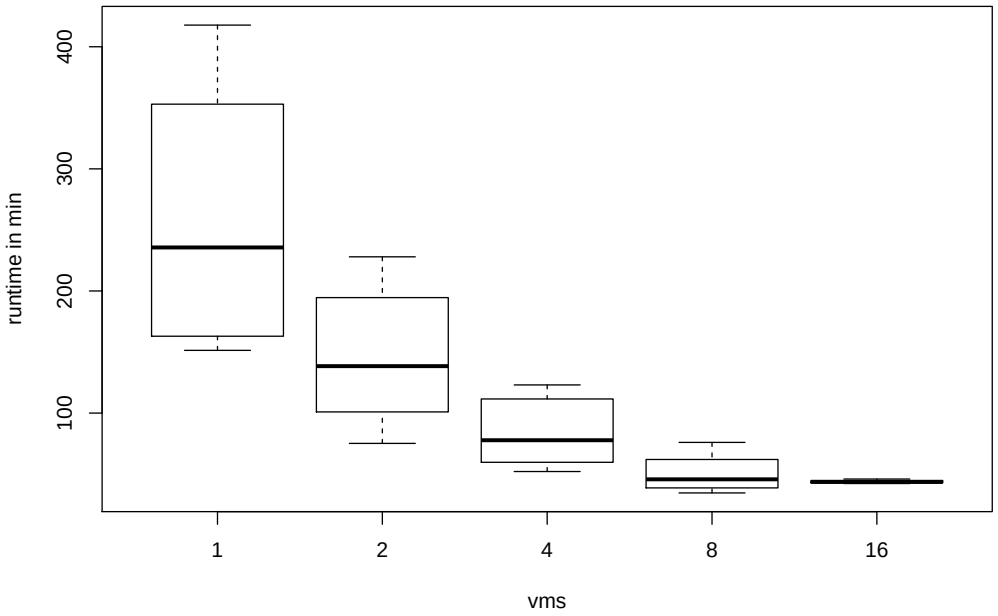


Figure 4.5: Box-Whisker-Plot with the number of VMs plotted against the measured runtime in minutes for the use case K-Means with an input size of 500 million points

measured values, which will be investigated in the next section.

Standard deviation of measurements

To describe different measurements within one function, a Linear Regression will be used. A basic requirement is a normal distribution of the underlying values. Therefore, in a first step the normal distribution of the measured runtimes is assumed. This means that all measurements of the same configuration, the same value of the introduced dimensions *number of VMs*, *CPUs per VM*, *RAM per VM* and *input size*, are normally distributed. Hence, each of these series of measurements has an error, which will be determined and valued with the standard deviation of these single measurements.

The estimation of this standard deviation, also called the *mean squared error of individual measurements*, is defined, as all other definitions in this section from [BHL+12], with the formula

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

where N describes the number of measurements, x_i the measured value and $\bar{x}$ the mean measured value.

It is calculated for each single series of measurement. Afterwards, these deviations are assumed to be normally distributed again, thereby the standard deviation of all series of measurements and so of the application's measurements can be obtained. Therefore, the arithmetic mean $\bar{v}$ and the standard deviation of all single standard deviations are calculated.

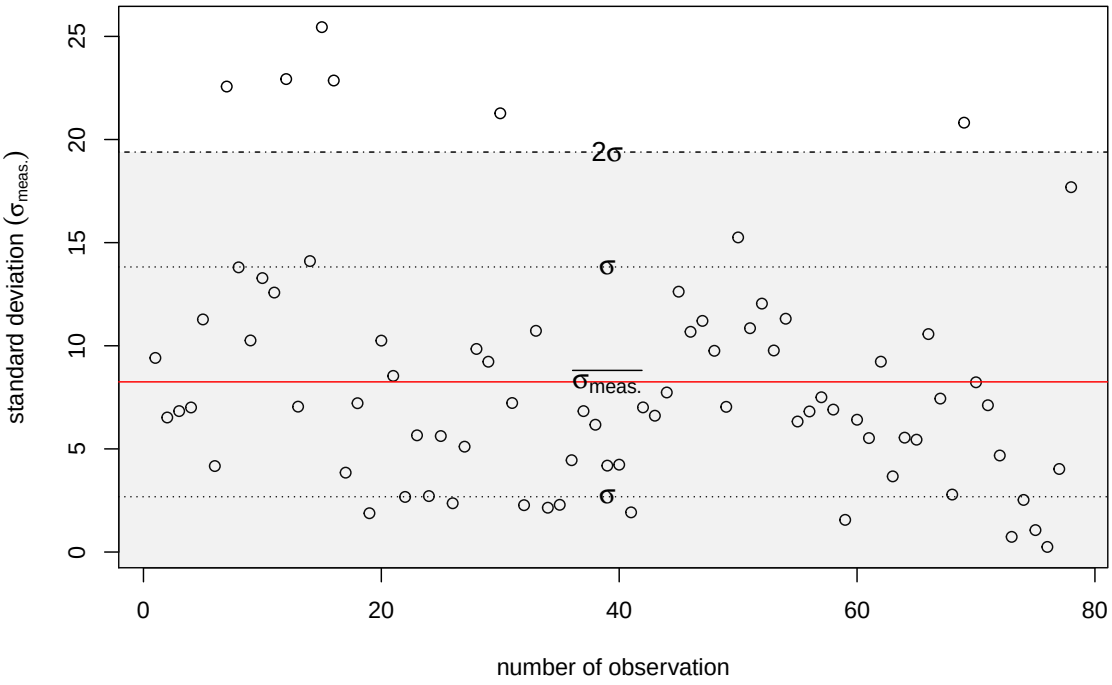


Figure 4.6: Standard deviation and mean of all measurements, use case K-Means

Following [BHL+12] also the error of the arithmetic mean of any series of measurement also needs to be considered. The resulting estimator of the standard deviation and mean are defined by:

$$\sigma_{series} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(\sigma_{measure,i} - \bar{\sigma_{measure}})^2} \tag{4.1}$$

$$\bar{v} = \bar{\sigma_{measure}} \pm \frac{\sigma_{series}}{\sqrt{N}}$$

To be certain that all values fall into a specified interval with a confidence of 0.95, the arithmetic mean needs to be extended by twice the standard deviation. This leads to the final formula:

$$\sigma_{min} = \bar{v} - 2 \cdot \sigma_{series} \qquad \sigma_{max} = \bar{v} + 2 \cdot \sigma_{series}$$

The σ_{min} is not interesting in this case, because the aim is to estimate the maximum deviation. So σ_{max} will be used to estimate the quality of the series of measurements.

In the end, the Linear Regression will calculate a function that will enable prediction of non-measured runtimes. These predictions, of course, will be described within a prediction interval, which will be larger the more error-loaded the original data is. To qualify the error of the regression the error of all series of measurements has to be taken into account.

In Figure 4.6 the standard deviations of all single measurements of the application K-Means are plotted. Additionally, the standard deviation of all single standard deviations is

plotted: the arithmetic mean of standard deviations as a red line and twice the standard deviation in both directions within the grayed block.

For the pictured example K-Means the normalized and so percentage values are

$$\bar{v} = 0.0825 \pm 0.0063 \qquad \sigma_{series} = 0.0557 \qquad \sigma_{max} = \bar{v} + 2 \cdot \sigma_{series} = 0.2002.$$

This leads to a maximum deviation of 20.02 %, which will be especially taken into account when validating the regression model in section 4.5.

The deviation of these measurements is very high, which has two reasons. On the one side, as could be already seen during the implementations of the measurements, the runtime values of the same configuration are wide spread. This is caused by the execution environment, the LRZ Compute Cloud. Its configuration of the nodes is very different and the load on the cloud is also very variable and therefore influencing.

Such a discrepancy was even observed in short runs under one hour. However, when extended to large measurements of over 24 hours, it had even more impact on the result. Thus, the worst absolute example had a lowest runtime of 21:09 hours, but a maximum runtime of 24:24 hours, a difference of 3:15 hours. Of course, this is better than the worst short run with a minimum of 0:14 hours and a maximum of 0:24 hours, regarding the relative deviation.

On the other side, the deviation depends on the number of measurements (compare equation (4.1)). This number is low, because the overall setup of this thesis was delayed due to a long debugging phase at the beginning when implementing the Flink environment and the measurement automating. Hence the measurements could not be done in the desired range due to these time constraints.

So this use case has to deal with such a high deviation, which will also influence the prediction out of the Linear Regression, but of course not the methodology itself. It will be suitably independent of the exemplary taken measurements.

4.2.3 Dataset definition

The values gained by the measurements are not passed completely to the function analysis. For each identical configuration setup the processed runtime is defined by the arithmetic mean of all measured runtimes. All these averaged runtimes are the basis for the calculation of the standard deviation of the complete series of measurements shown in section 4.2.2.

The resulting dataset is not normalized and consists besides the double runtime-means only of integer values. An excerpt of values of the application K-Means is shown in table 4.3.

runtime	vms	cpus	ram	input
175323.625	8	8	4	5e+07
2740305.500	4	1	8	20e+07
43684560.000	8	1	8	500e+07

Table 4.3: Dataset excerpt K-Means as passed into the MLR.

For the application K-Means 450 single measurements were added to the initial data set. Two third of these runtimes were measured at the input sizes 5e+07 and 500e+06 as detailed in table 4.2. For the biggest input only 23 measurements could be taken due to

time constraints at the end of the thesis. They were averaged for the same configurations to 69 observations, which build the final basis for the regression model.

The runtime value is given in milliseconds. For the purpose of legibility this dimension is reduced to minutes by division of $1000 \cdot 60 = 3600$ for all plots. This mathematical transformation has no influence on the statistical validity and can hence be applied.

4.2.4 Adjusted R^2 value

The so-called R-squared (R^2) value is a statistical measure of how close the observed data corresponds to the fitted regression line. It is also known as the coefficient of determination. For a multiple regression the responding value is the *multiple* R^2. Its value will be used to evaluate a regression model with a corresponding regression formula. The better it fits to the given observations, the closer the value is to 1.

The difference between the predicted value and the actual value of the original data is called the residual. The Residual Sum of Squares (RSS) for a single prediction variable, the Mean Sum of Squares (MSS) and the resulting multiple R^2 are given by

$$MSS = \sum_{i=1}^{n} (y_i - \bar{y})^2 \qquad R^2 = 1 - \frac{RSS}{MSS}$$

where y_i is the dependent variable observation, $\hat{y}_i$ its prediction from the regression model and $\bar{y}$ the mean of the observations.

In order to define a comparable value over different linear regression models it is necessary to involve the number of degrees of freedom (prediction variables) and the quantity of measured data points. Therefore, the *adjusted* R^2 refines the multiple R^2 and is defined as

$$R^2_{adj.} = 1 - (1 - R^2)\frac{n-1}{n-p-1}$$

where p is the number of prediction variables without the intercept and n is the amount of data points, the so-called observations. The thereby gained quality value is comparable to other Linear Regressions, although the dimensionality is different. $\frac{n-1}{n-p-1}$ in the upper equation penalizes a higher number of independent variables in comparison to the observations and lowers the multiple R^2 value.

4.2.5 Dimensions' transformations

To perform a MLR on the complete dataset it is necessary to define the transformation of the independent dimensions *number of VMs*, *CPUs per VM*, $CPUs \cdot VMs$, *RAM per VM* and *input size* in relation to the dependent dimension *runtimes*. These transformations will be described by a mathematical figure $f_i : x_i \mapsto y_i$ for $i = 1,..,7$, which will be inserted in equation (4.6) as the functions $f_1, \ldots, f_7$. The better it fits, meaning the higher the $R^2_{adj.}$ value is, the better the data points are mapped. Hence the aim is to find the best fitting transformation for each dimension.

When plotting such a two-dimensional (2D) correlation it might be possible to define the function simply on closer inspection, but it is recommended to determine this with an algorithm comparing different $R^2_{adj.}$ values and take the highest as the resulting function.

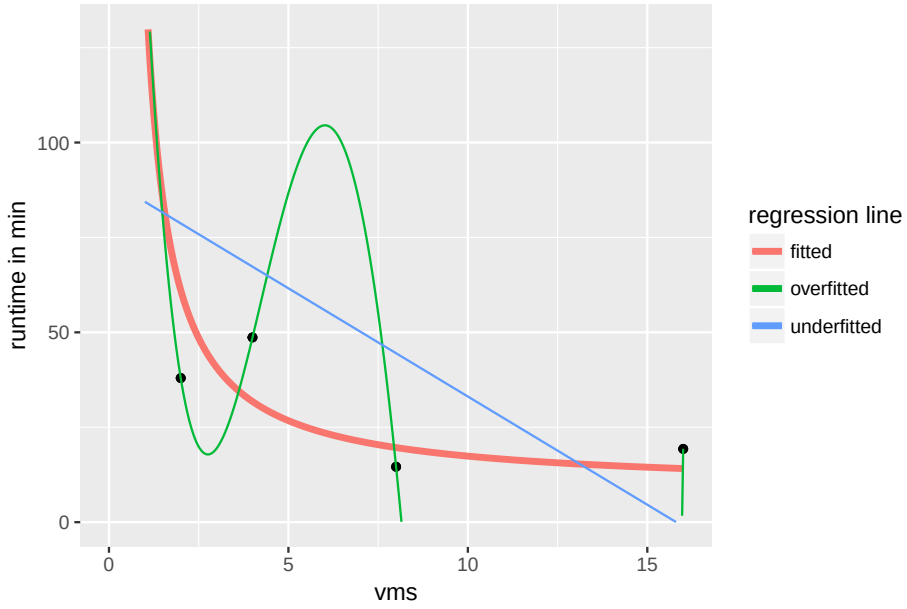


Figure 4.7: Regression line plot with overfitting and underfitting

Therefore, the dataset will be separated into five 2D-datasets with the dimension *runtime* and in each case one of the dependent dimensions. The resulting datasets are shown in table 4.4.

(a)	runtime / vms	(b)	runtime / cpus	(c)	runtime / ram	(d)	runtime / input
(e)	runtime / $cpus \cdot vms$	(f)	runtime / $cpus \cdot ram$	(g)	runtime / $vms \cdot ram$		

Table 4.4: 2D-datasets to determine the dimensions' transformations

On these datasets a simple Linear Regression will be applied. The possible resulting figures are limited by the objective to derive a possibly high generic regression function. Any overfitting shall be excluded as well as possible.

Overfitting occurs when the regression model is too complex, for example by having too many independent parameters for the amount of information in the datasets [HJ15]. In that case, the $R^2_{adj.}$ value would be exaggerated because the fitting is too accurate to the observed data. Hence all new prediction would be extremely defective. Figure 4.7 shows three different regression lines, while the red one is the accurately fitted, the green one is overfitted and the blue one is underfitted.

For the plotted graphs, the $R^2_{adj.}$ values and the corresponding regression function are listed in table 4.5.

To avoid this effect, the parameter of each dimension is limited to one transformation instead of a linear combination of several. Also the range of transformations is limited, especially any polynomial term is not considered as generalizable and therefore discharged.

regression function	$R^2_{adj.}$	fitting
$runtime \sim\ vms + vms^2 + vms^3 + vms^4$	1.000	overfitted
$runtime \sim\ \exp(\frac{1}{vms})$	0.929	fitted
$runtime \sim\ vms$	0.375	underfitted

Table 4.5: Regression function with overfitting.

The selected seven possible transformations are as follows:

$$y \sim\ x \qquad y \sim\ \exp(x) \qquad y \sim\ \exp\left(\frac{1}{x}\right)$$

$$y \sim\ \frac{1}{x} \qquad y \sim\ \exp(-x) \qquad y \sim\ \ln(x)$$

$$y \sim\ \sqrt{x}$$

With all these functions a simple Linear Regression is done with all dimensions of the problem space. For each try the $R^2_{adj.}$ value is computed. The function with the highest value will be the chosen one for this dimension. If the $R^2_{adj.}$ values are equal, the $F-statistic$ is used to determine the transformation.

Algorithm 1 Selection of dimensions' transformations

```
1:  procedure DimensionsFunction(runtimes, dim)
2:      F ← {dim, √dim, 1/dim, ln dim, exp dim, exp −dim, exp 1/dim}
3:      R²_adj. ← −∞
4:      F_stat. ← −∞
5:      for all f ∈ F do
6:          model ← MLR(runtimes ∼ f)
7:          if R²_adj.(model) > R²_adj. then
8:              result ← f
9:              R²_adj. ← R²_adj.(f)
10:             F_stat. ← F_stat.(f)
11:         else if R²_adj.(model) = R²_adj. then
12:             if F_stat.(model) > F_stat. then
13:                 result ← f
14:                 R²_adj. ← R²_adj.(f)
15:                 F_stat. ← F_stat.(f)
16:             end if
17:         end if
18:     end for
19:     return result
20: end procedure
```

The *f-statistic* is a measurement for the existence of any linear association between Y and *any* of the X variables, following [She09], and therefore if at least one of the coefficients is

nonzero. It is defined by

$$F_{stat.} = \frac{MSS - RSS}{p} \cdot \frac{n - p - 1}{RSS}$$

where p is the number of prediction variables without the intercept and n is the amount of data points.

The complete selecting process is shown in algorithm 1.

To illustrate this process the resulting plots of the resulting Linear Regression curves for all dimensions are shown in figure 4.8. In each plot all measurements are plotted, each independent dimension against the runtime in minutes. The red curves correspond to the selected, best fitting transformations from the described analysis. The other curves are plotted in contrast with their transformations in the legend, limited to two lines to not reduce the visibility. The analysis bases on the complete K-Means dataset described before. Due to the fact that the dataset consists of many distributed measurement values, which correspond to different values of the other dimensions, the regression lines do not always clearly look like the most significant. Hnec this is a good example, why the function cannot be defined only on closer inspection. It is absolutely necessary to look at the $R^2_{adj.}$ values.

Decided by algorithm 1 the best transformations for the independent dimensions in the use case are

$$\begin{aligned}
f_{vms} &: y \mapsto \frac{1}{x} \\
f_{cpus} &: y \mapsto \exp \frac{1}{x} \\
f_{ram} &: y \mapsto \exp \frac{1}{x} \\
f_{input} &: y \mapsto \sqrt{x} \\
f_{cpus \cdot vms} &: y \mapsto \exp \frac{1}{x} \\
f_{ram \cdot vms} &: y \mapsto \exp -x \\
f_{cpus \cdot ram} &: y \mapsto \exp -x
\end{aligned}$$

If two or more transformations would have the same maximum $R^2_{adj.}$ value, the optimal transformation is determined by the highest f-statistic value. It is a good indicator of whether there is a relationship between the predictor and the response variables. The further the f-statistic is beyond 1 the better it is.

All these transformations describe the best fitting estimation for all observations in each dimension. In the following sections they will be used to determine on the one hand the statistical significance of each dimension and on the other hand, if added to the Linear Regression due to significance, the estimators of the resulting regression coefficients of each dimension with which the runtime will be predicted.

4.2.6 Determining Statistical Significance of Dimensions

In the previous sections, the problem space and its dimensions were defined, as well as the appropriate transformations. Before the calculation of the Multiple Linear Regression can be

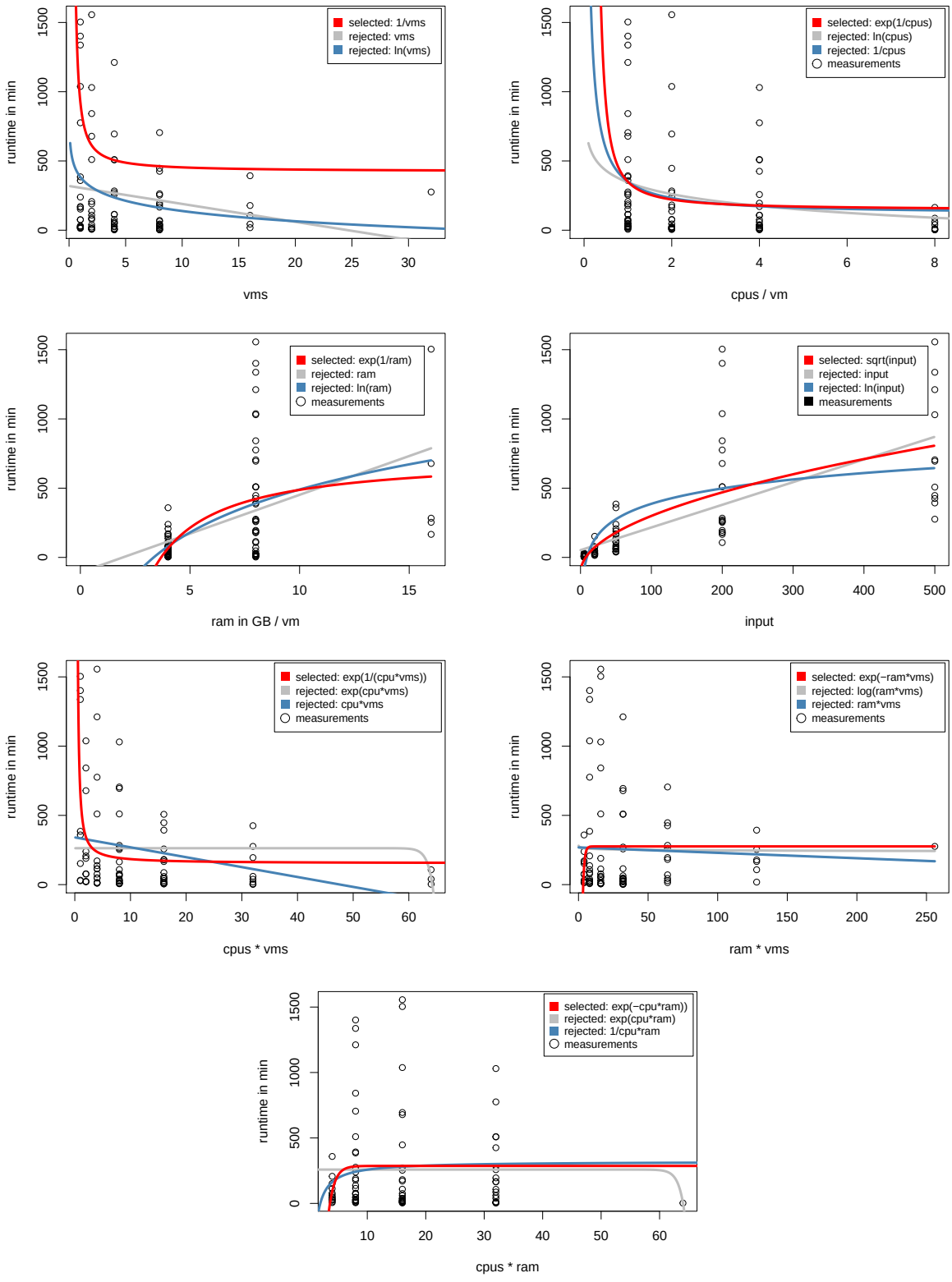


Figure 4.8: Dimension-wise regression functions and transformations

initialized, the dimensions need to be tested for significance. This means that it is necessary to determine if any transformed parameter improves the quality of the regression and the fitting itself or if it is statistically insignificant and has no additional benefit. This section will identify the best predicting model.

The already explained $R^2_{adj.}$ value is one measurement that can be used. Thus, all possible combinations are iterated in ascending order. First the simple regression function $t \sim 1$ is tested and afterwards all other dimensions and their combinations. To do so 2^n tests are necessary, where n is the number of independent dimensions. In the use case K-Means, it would be required to test seven dimensions in $2^7 = 128$ tests. From all of them the $R^2_{adj.}$ value would be compared and the combination with the highest value would be the optimal formula.

Besides this naive approach more complex statistical methods exist, which reduce the executed amount of tests. Hence, the methodology will introduce the *Akaike's Information Criterion* (AIC) and the *Bayesian Information Criterion* (BIC).

An important instrument in both criteria is the likelihood function defined in [Rei11]. It is proportional to the density function for a random variable to be observed in a statistical model. Within the given parametric model, the likelihood function

$$L(\theta) = L(\theta; y)$$

measures the relative plausibility of various values of θ for a given observed data point y. The values of the likelihood function are sometimes standardized by the maximum value of it, the *maximum likelihood estimators*. For calculations purpose, instead of the likelihood function its natural logarithm, the *log-likelihood function*, is used:

$$l(\theta) = \ln\left(L(\theta; y)\right).$$

The AIC, as well as the BIC, are based on this log-likelihood function. Both definitions are derived from [She09]. AIC tries to balance the degree of accuracy of the fit with a penalty for the complexity of the model. As defined, the smaller the value of AIC the better the model tested.

The formula used to describe the calculation of the AIC in this thesis is derived from the statistical program R, which is uses

$$AIC = -2 \cdot l(\theta) + 2 \cdot (p + 2)$$

where p is the number of prediction variables, adding two for the intercept and the error. In literature other definitions can be found, but since the exemplary calculation is done in R, also its definition is taken. $2 \cdot (p + 2)$ is called the penalty term which increases the AIC the more independent variables are in the model. It is generalized as $k \cdot (p + 2)$, where k is the penalty factor. On the one hand, interpreted from [Aka11], models which are too simple to adequately fit the data will be characterized by a large fit accuracy term due to small penalty terms. On the other hand, models that correspond well with the data, but do this with unnecessary parameters, will be characterized by small fit accuracy terms due to large penalty terms. Models which provide an eligible balance between correctness to the data and parsimony of parameters correspond to small AIC values due to the sum of the two components.

The bayesian approach BIC is in [She09] also defined as the smaller the value the better the model. It is derived from the general AIC definition with a penalty factor of $k = \ln(n)$ instead of $k = 2$ in the AIC:

$$BIC = AIC_{\log n} = -2 \cdot l(\theta) + \ln(n) \cdot (p + 2) \tag{4.2}$$

where n is the amount of data points. For $n > 8$ ($\log n > 2$) the penalty term in BIC is greater than in AIC. Hence BIC penalizes complex models more heavily and prefers simpler models in comparison to AIC.

As stated in [She09] there is no clear choice between AIC and BIC, but regarding the sample size, a trend is describable. Because BIC is asymptotically consistent as a selection criterion, the probability that BIC will identify the correct model in a family of models (including the best predicting model) approaches one, as the sample size increases with $N \to \infty$. AIC tends to choose too complex models for $N \to \infty$. However, for finite samples, BIC often chooses too simple models, because of the strong penalty on complexity. The best way is to consider both, weight the benefits against each other, and especially review the observation size.

Besides these possibilities, more exist like for example a weighted likelihood method, but this thesis is confined to AIC and BIC. An efficient approach to analyze the regression function for the best predicting model can be done with R. Therefore it is only necessary to define a maximum and minimum formula and execute the AIC or BIC method. The Linear Regression Model is instantiated with the R-command `lm(formula)`. The following example contains all eight transformed dimensions, the measured runtime dependent on the four configurable dimensions and the three combined ones:

Listing 4.1: BIC for the best predicting model in R

```
formula_min <- formula("runtime ~ 1")
formula_max <- formula(runtime ~ I(1/vms)+sqrt(cpus)+
        exp(1/ram)+sqrt(input)+exp(1/I(cpus * vms))+
        exp(-I(ram * vms))+exp(-I(cpus * ram)))

bestmodel <- step(lm(formula_min, ...),
        scope = list(lower = formula_min, upper = formula_max),
        direction = "both", criterion = "BIC", k=log(n))

getCall(bestmodel)
```

Here a BIC with a $k = \log(n)$ weighted penalty term is determined in `both` directions, which means that R starts with the minimal formula and ends with the complete stated regression formula and the other way around. In each step the BIC value is calculated and the parameter with the lowest BIC under the previous BIC is permanently added. Because the BIC is a modulation of the AIC (compare equation (4.2)), it is still stated as AIC in the output of R:

Listing 4.2: BIC step to determine next added dimension

```
Step:  AIC=2779
runtime ~ sqrt(input) + exp(1/I(cpus * vms))

                        Df Sum of Sq       RSS  AIC
+ I(1/vms)               1  1.03e+15 1.15e+16 2774
+ exp(-I(cpus * ram))    1  4.94e+14 1.20e+16 2777
+ exp(-I(ram * vms))     1  3.42e+14 1.22e+16 2779
+ sqrt(cpus)             1  3.30e+14 1.22e+16 2779
<none>                              1.25e+16 2779
+ exp(1/ram)             1  1.30e+13 1.25e+16 2781
- exp(1/I(cpus * vms))   1  6.95e+15 1.94e+16 2814
- sqrt(input)            1  2.37e+16 3.62e+16 2867
```

In a previous step `sqrt(input) + exp(1/I(cpus * vms))` were already added to the best predicting model marked with "`-`". In this step, `exp(1/ram)` will also be added, because it has the lowest AIC value over the "`<none>`" value. Continued to the end, this algorithm exposes its best model as:

$$\text{runtime}_{\text{BIC}} \sim \frac{1}{vms} + \sqrt{input} + \exp\left(-ram \cdot vms\right)$$

As mentioned before, the BIC method tends to choose a simple model, because of the heavy penalty on complexity compared to the sample size. Running the same configuration with the AIC method, the best predicting model is:

$$\text{runtime}_{\text{AIC}} \sim \frac{1}{vms} + \sqrt{input} + \exp\left(-ram \cdot vms\right) + \exp\left(\frac{1}{cpus \cdot vms}\right) \tag{4.3}$$

Because the sample size in the use case is finite, the AIC is chosen and the best predicting model would be in equation (4.3). In this case, the BIC would completely remove the dimension *cpus*, which would be an undesired effect, because the number of CPUs had an obvious effect on the runtime, although not significant enough for the BIC with this amount of the sample size.

In the AIC result, both *cpus* and *ram* are also not included in their own dimension, only in a combined one. That these dimensions did not primarily influence the runtime was already seen during the measurements and was expected. That for example the dimension *cpus* in combination with *vms* is significant for the AIC is caused by the fact that the contributing Flink task managers are defined by the sum of all CPUs per VM and it is therefore an important value.

However, if one dimension would be completely removed from the regression model, for example by applying the stricter BIC on the problem space, thereby also the data set would change, which will be explained in the next section.

4.2.7 Problem Space Refinement

With the determinations in the last section, the significant dimensions are exposed. With these the main parameters of the Multiple Linear Regression are obtained.

If a dimension would be completely rejected, in the transformation of their single dimension and in all combined ones, it would be necessary to alter the problem space and with it the

runtime	vms	CPUs	ram	input
6696720.9	4	4	8	5e+08
6838352.2	4	2	8	5e+08
16354774.5	8	2	8	2e+09
15204575.0	8	1	16	2e+09

→

runtime	vms	ram	input
6767536.5	4	8	5e+08
16354774.5	8	8	2e+09
15204575.0	8	16	2e+09

Table 4.6: Dataset dimensions' reduction.

data base. Therefore it would be necessary to recalculate the arithmetic mean of the runtime dimension. As defined in section 4.2.3 the measurements are not passed as single observations into the regression, but as the arithmetic mean runtime of the same configuration.

For this reason, if a dimension is dropped, for the same values of the remaining dimensions the mean of the single runtime values needs to be recalculated. Regarding the use case, by using the BIC the dimension *cpus* would be removed as shown in table 4.6.

Afterwards the methodology described so far would have to be applied again regarding all resulting values: the standard deviation of measurements (section 4.2.2), the statistical significance of the dimensions (section 4.2.6) and the determination with the AIC.

4.3 Function Analysis

With the defined and discussed problem space dimensions as well as the gathered runtime values of the measurements, the main calculation can be started. This section will present the methodology from processing the dimensions and values through a Linear Regression to obtain a high dimensional function. This shall be as generic as possible, but still predict runtimes as accurately as possible. These two objectives need to be weighed up against each other to find a balance: generic enough to be comparable to other applications, but of course accurate enough to predict the runtime within an acceptable error tolerance.

With the resulting function it will be possible to predict any runtime value of any cloud configuration covered from the problem space. In this thesis, a multivariable Linear Regression will be used. As stated in [HJ15], such models are used extensively in almost all scientific areas, like medicine, ecology, biostatistics and many others, and are highly reliable under the discussed circumstances.

4.3.1 Regression

Regression analysis is a statistical approach with a set of statistical processes for estimating the relationships between variables. The regression model is stated in terms of a weighted sum of a set of independent variables to predict a response (dependent) variable. The overview in this section based on the findings of [HJ15].

The Simple Linear Regression models the relationship between two variables Y and X. Then, a specific value of X will predict the value of Y. Mathematically described, the regression of a random variable Y on a random variable X is

$$\mathrm{E}(\mathrm{Y} \mid \mathrm{X} = x)$$

where E is the expected value of Y with the specific value x of X. Such a predictive regression model can be notated with all factors as:

$$\mathrm{Y} = \mathrm{E}(\mathrm{Y} \mid \mathrm{X} = x) + \varepsilon = \beta_0 X_0 + \beta_1 X_1 + ... + \beta_p X_p + \varepsilon$$

where Y is the response variable, $X_1, ..., X_p$ are the predictor variables, β_0 is an optional intercept parameter, $X_0 = 1$ is considered and $\beta_1, \cdots, \beta_p$ are the weights or regression coefficients corresponding to $X_1, \cdots, X_p$. The random error term ε is necessary, because there will be some variation in Y due strictly to random phenomenon that cannot be predicted or explained, as defined by [She09]. ε has a mean of zero and a variance of σ^2. So all unexplained variations are called the random error, but ε does not depend on X, nor does it carry any information about Y.

The same equation can be written in matrix notation, which will be used in the remaining thesis:

$$\mathrm{Y} = \mathrm{X} \cdot \beta = \begin{bmatrix} 1 & X_1 & \cdots & X_p \end{bmatrix} \cdot \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_\mathrm{p} \end{bmatrix} + \varepsilon \tag{4.4}$$

The shown example is a Multiple Linear Regression (MLR) model, which means that the response is a linear combination of more than one parameter, the regression coefficients $\beta_0, \cdots, \beta_p$ and the predictor variables $X_1, \cdots, X_p$. A simple Linear Regression would only imply one single scalar predictor variable and one single scalar response variable. It is assumed that the variables are normally distributed:

$$\mathrm{VAR}(\mathrm{Y} \mid \mathrm{X} = x) = \sigma^2.$$

where the variance can be written as the squared standard deviation detailed in section 4.2.2.

For both variables it is important that only the coefficients have to be linear. The variables themselves are processed as real numbers and can also be mathematically transformed to any arbitrary type of function. This can be for example an exponential (e.g. $\exp x$), logarithmic (e.g. $\log x$) or polynomial (e.g. $x + x^2 + x^3$) transformation. Therefore the Linear Regression is very powerful while covering all possible linear combinations of mathematical terms. The main aim is to determine the coefficients above, which is called fitting the data by the model. One approach is the *ordinary least squares (OLS)* method.

In OLS the objective is to minimize the sum of the squared differences between the observed responses (initial values in the given dataset) and the predicted values from the linear regression function: the residual sum of squares (RSS) shown in equation (4.5). The smaller the difference is, the better the model fits the data.

$$RSS = \sum_{i=1}^{n} (y_i - \widehat{y_i})^2 \tag{4.5}$$

Another approach is to use weighted least squares, where the initial datasets can be weighted if they are not equally reliable. Other, more general models also exist, but as a normal distribution for the observations is assumed they were not considered.

Therefore, the Linear Regression will be used to estimate the function of prediction. Its definition and derivation will be presented in the next sections, resulting in the prediction formula in section 4.4.2.

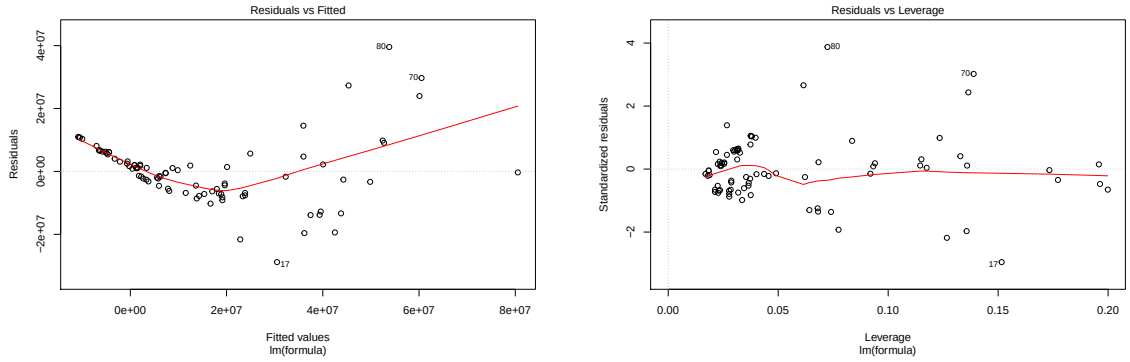


Figure 4.9: Outlier and Leverage points

4.4 Multiple Linear Regression Model

The selected regression model is used to estimate the coefficients $\beta_0, \cdots, \beta_p$ of the regression formula. The response variable Y is the runtime t and the prediction variables are the configuration dimensions determined in section 4.2.6: vms, $input$, $ram \cdot vms$ and $cpus \cdot vms$. Applying them to the equation (4.4) the initial formula for the analysis is:

$$runtime \sim \begin{bmatrix} 1 & \frac{1}{vms} & \sqrt{input} & \exp\left(\frac{1}{cpus \cdot vms}\right) & \exp\left(-ram \cdot vms\right) \end{bmatrix} \cdot \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \end{bmatrix} + \varepsilon \qquad (4.6)$$

To determine the quality of a regression the adjusted R-square value will be used.

4.4.1 Outlier and Leverage point treatment

Each data point included into the regression analysis has an influence on the result. The influence differs from point to point, but all should to be within a certain range. To identify potentially misleading values, for example due to a huge measurement error, it is convenient and necessary, as for example described in [She09], to identify *Leverage points* and *Outliers*.

Hence, a Leverage point is an observation of the regression model which independent values have an unusually large effect on the estimated regression model. Outliers are points that do not follow the pattern set by the bulk of the data.

Outliers can be easily identified by plotting the residuals or listing them in a table. Figure 4.9 shows the residuals of the use case K-Means, in which the points 70 and 80 are marked as Outliers, and the points 17, 70 and 80 as Leverage points.

In the left figure the fitted values of the model are plotted against the residuals. The red line shows the average of the residuals at each value of the fitted points. In an optimally balanced fitting Linear Regression between the observations that red line would be horizontal. The right plot shows the leverage to the standardized residuals, which is the residual divided by its standard deviation, and again the average as a red line. This red line is close to an optimal balance of the observations, although the model is driven by some spikes.

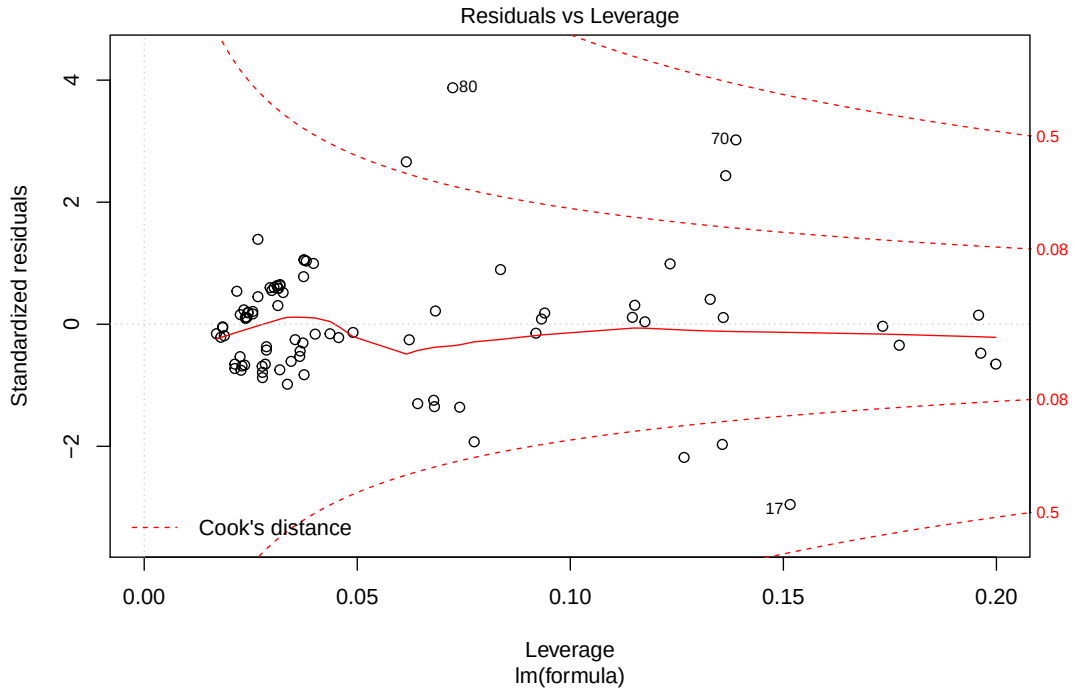


Figure 4.10: Cook's distance plot for observations in the use case K-Means

The more the points are displayed at the right end of the horizontal scale, the more influence these points have on the model. Additionally, the more the points are at the top or bottom of the vertical scale, the further is their distance from the regression function. Very important to consider are the points in the upper right or bottom right corner.

However, because these values are hard to interpret on their own, a combined value is generally accepted nowadays: Cook's distance. Detailed in [Coo11], it is a means to value the influence of single observations on the linear regression model. Cook's distance is based on the idea of contrasting the results of a regression with and without an observation, a data point.

Cook's distance is not a rigorous criterion for automatically accepting or rejecting cases. It only indicates an anomalous behavior not according to the rest of the model, or it can indicate the most important case in the analysis that may not be represented by the other data points.

Important is that the Cook's distance does not distinguish these possibilities, so all cases with relatively huge values need to be considered manually whether they can be excluded from the data set or whether even the model needs to be refined. Cook recommends to test all identified points whose distance is relatively high or over the value of 0.5. Cook's distance D_i of observation i (for $i = 1, \ldots, n$) is defined for R[8] as the sum over differences in the regression model if observation i is removed from the model by

[8] `http://r-statistics.co/Outlier-Treatment-With-R.html`

$$D_i = \frac{\sum_{j=1}^{n} \left(\hat{y}_j - \hat{y}_{j(i)}\right)^2}{(p+1) \cdot \frac{RSS}{n}}$$

where $\hat{y}_j$ is the value of j-th fitted response when all the observations are included, $\hat{y}_{j(i)}$ the value, where the fit does not include observation i, RSS is the residual sum of squares defined before and n the sample size, p is again the number of prediction variables, here adding one for the intercept.

With this definition, Cook's distance can be included into the outlier detection. The distance values can be gained manually in a table again, or with R, which is used here. In figure 4.10 the Leverage is plotted against the Standardized residual, and the Cook's distance is drawn as a dashed red line. One line for the value 0.5 suggested by Cook in [Coo11] and one with the in this literature named value of four times the mean of all distances, $4 \cdot \bar{D} = 0.08$ for the mentioned use case.

If there are one or more points over the value of 0.5 or relatively far away from all others, then these points need to be investigated, starting with the most divergent. If the value of such a point is considered relevant and the measured runtime is plausible, it may not be removed from the dataset. The Outlier and Leverage point detection cannot be misused as *cherry picking* to fit the model better to the data. Usually all measurements are a part of the model and none should be removed. Nonetheless if measurements show an enormous deviation in their single values this observation is influenced by abnormal loads or network traffic in the cloud. Then, and only then, may such a point be removed.

For all other detected points which are not removed, the model could not be completely sufficient in the regression. Any questioning if this regression model is insufficient is declined for this methodology, because it is considered strong and generic enough to deal with these possible outliers which are a part of a desired effect. Therefore, it can be stated again, that the quality of the regression model turns on the quality of the measurements.

However, if the outlier is considered defective with the underlying single measurements, the data point should be removed from the data set. Afterwards the model would not have to be processed from scratch again, because here the already determined model is optimized and not challenged.

In the use case all observations are under the 0.5 value of Cook's distance. Although they are marked as an Outlier respectively as a Leverage Point in the plot due to a relative high distance to the average, they may not be removed. All these corresponding runtimes belong to measurements of a smaller input size at only one VM. These runtimes might be not totally according to the rest of the observation, but the measured runtimes are reasonable and may never be removed.

This would affect the regression model. To prevent this, no observation is removed and all are finally passed to the estimation of the regression coefficients.

4.4.2 Linear Regression Coefficients

After the outlier treatment all preliminary work is done to calculate the coefficients of the Multiple Linear Regression. With the determination of the AIC the final regression formula was found for the use case K-Means:

$$runtime \sim \begin{bmatrix} 1 & \frac{1}{vms} & \sqrt{input} & \exp\frac{1}{cpus \cdot vms} & \exp -ram \cdot vms \end{bmatrix} \cdot \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \end{bmatrix} + \varepsilon \tag{4.7}$$

As stated by Brian Caffo in [Caf15], linear regressions and especially for multivariate regression models are never fitted manually in practice, but with software like R. However, the general idea can be described. The ordinary least squares solutions needs to estimate the values of the coefficients $\beta_0, \cdots, \beta_p$. These estimates are the values of $b_0, \cdots, b_p$ for which the residual sum of squares defined in [She09]

$$RSS = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} (y_i - b_0 - b_1 x_{1i} - b_2 x_{2i} - \cdots - b_p x_{pi})^2 \tag{4.8}$$

is minimized. This results in a system of $p+1$ equations and $p+1$ unknowns and should not be solved without a computer program. With the support of R the regression model is set and the linear regression is applied as shown in Listing 4.3.

Listing 4.3: Multiple Linear Regression command in R

```
model = lm("runtime ~ sqrt(input) + exp(1/I(cpus * vms))
        + I(1/vms) + exp(-I(ram * vms))", data)
```

There the formula is describe with the dimensions stated in equation (4.7). The `lm` function defines an OLS Linear Regression with the dataset `data` and is addressable with the variable model. This function calculates the estimators of the regression: the coefficients. Beside them other important values are provided. The complete output is shown in listing 4.4.

Residuals

Residuals are described with their mean and extreme values. In an optimal case, the median should lay at zero, here it is shifted to negative values. Due to the definition the value is calculated with $y_i - \widehat{y_i}$, which means, negative residuals are fitted with a too high value. In the use case negative residuals result in too long runtimes compared to the observations. Following the model output, the majority is fitted with too long runtimes, while the maximum residual belongs to a fitted value of a much too short runtime. This data point 64 belongs to the longest measurement and was marked as an outlier, but considering that the leverage was not large, it was not removed from the dataset in the section before.

Listing 4.4: Multiple Linear Regression output from R

```
> summary(model)

Call:
lm(formula = "runtime ~ sqrt(input) + exp(1/I(cpus * vms))
        + I(1/vms) + exp(-I(ram * vms))", data = measures)

Residuals:
      Min         1Q     Median         3Q        Max
-28833089   -6191417    -320140    5413416   39556687

Coefficients:
                       Estimate Std. Error t value Pr(>|t|)
(Intercept)          -2.79e+07   3.91e+06   -7.12  4.1e-10 ***
I(1/vms)              3.38e+07   6.36e+06    5.31  9.4e-07 ***
sqrt(input)           7.86e+02   5.67e+01   13.87  < 2e-16 ***
exp(1/I(cpus * vms))  7.15e+06   3.54e+06    2.02    0.047 *
exp(-I(ram * vms))   -1.25e+09   2.65e+08   -4.70  1.1e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 10600000 on 80 degrees of freedom
Multiple R-squared:  0.788,     Adjusted R-squared:  0.777
F-statistic: 74.1 on 4 and 80 DF,  p-value: <2e-16
```

Regression Coefficients

The coefficients are the estimates of the $\beta_0, ..., \beta_p$ variables, where β_0 is named the intercept. The others describe the factors of the corresponding dimensions and together define the slope of the regression function. Their values can be found in the column *Estimate* and based on equation (4.7) the prediction function of this use case is defined by:

$$\mathrm{Y} = t_{runtime} = \begin{bmatrix} 1 & \frac{1}{vms} & \sqrt{input} & \exp\left(\frac{1}{cpus \cdot vms}\right) & \exp\left(-ram \cdot vms\right) \end{bmatrix} \cdot \begin{bmatrix} -2.79\text{e}7 \\ 3.38\text{e}7 \\ 7.86\text{e}2 \\ 7.15\text{e}6 \\ -1.25\text{e}9 \end{bmatrix} + \varepsilon \tag{4.9}$$

The influence of each dimension cannot be read from the values of the coefficients, because they deal with the absolute values of the data set and their transformation. For example the dimension *vms* is only $[\frac{1}{1} = 1.000; \frac{1}{32} = 0.031]$, but the range of the *input* size is $[\sqrt{5\text{e}7} = 7,071; \sqrt{5\text{e}9} = 70,711]$.

The `Std. Error` is an estimate of the standard deviation of each coefficient, the discrepancy between the fitted value of the response variable and the actually observed value.

The last two columns define the statistical significance of each dimension. The major idea following [Caf15] is to reject the null hypothesis, which states that each independent dimension has no effect on the dependent response. Therefore the `Pr(>|t|)`-value is a measurement whether any (linear) relationship exists or not.

According to [Pri17] the `t value` is the estimate divided by the standard error and is compared to the values in the *Student's t-distribution*, which describes the expected behavior of the mean of a certain number of observations. If 95% of this t-distribution are closer to the mean than the t-value of the estimated coefficient, the `Pr(>|t|)`-value is 0.05 (5%), which is meant to be the significance level. A Level of 0.05 or less is generally accepted to reject the null hypothesis.

As it can be seen in the model summary all dimensions influence the response variable and the null hypothesis can be rejected with a 99% confidence for the dimension $cpus \cdot vms$ and with a confidence of 99.9% for all others.

The Residual Standard Error

The standard deviation of the residuals is described as the `Residual standard error` and is a measure of the quality of the complete regression model. Because the regression formula defined before included an error term, it is not possible to predict the runtime perfectly. This given error is the average the response deviates from the observed perfect regression line.

The `degree of freedom` in statistics is the number of variables in the final calculation which are free to vary. It can also be described as the dimension of the subspace of predictive dimensions. In the case of a Linear Regression they are defined as the amount of observations, subtracted by the number of independent dimensions and one for the intercept:

$$\mathrm{df} = n - p - 1.$$

In this use case the degrees of freedom are $85 - 4 - 1 = 80$.

Qualitative Regression Measurements

The `Multiple` and `Adjusted R-squared` values were already explained in section 4.2.4. The $R^2_{adj.}$ has the final value 0.777. Its prediction capability will be evaluated in the next chapter 5.

The `F-statistics` described in section 4.2.6 is an indicator if any relationship between the prediction and the response variables exists or not. With a value of 74.1 as shown in listing 4.4 a relationship is assumed. The final `p-value` is equivalent to the defined `Pr(>|t|)`-valued, but for the entire regression model. The value indicates a very high significance, so the model and the obtained results can be stated significant for the given data set.

4.4.3 Linear Regression Representation

The specified coefficients are estimated by the regression model and the estimated runtime is given in equation (4.9).

Results of a five-dimensional space are very difficult to plot clearly arranged. To solve this problem, the sample space is split and only a subset is displayed. In each plot in figure 4.11 and 4.12 one independent dimension is plotted against the dependent runtime. The observations are inserted with their error range derived from the standard deviation of all measurements. The second dimension splits these observations at two distinct values, the lower one in the left part, the higher one in the right part.

The third dimension colors all observations according to their values. The last variable is fixed. In most cases, this is the value for $ram \cdot vms$. In addition, the resulting regression

curves are indicated with two supplementary fixed values of the third, coloring dimension. An overview is given in table 4.7.

dimension	identification
runtime	y-values
1st	x-values
2nd	split (left, right)
3rd	color
4th	fixed value

Table 4.7: Dimensions' features of regression plots in figure 4.11 and 4.12.

The runtime and input values for all plots and the underlying regression model were transformed to be readable. The runtime is given in minutes and the input size in 10^7 (1e7) points. To all regression curves an uncertainty should be added, which was not possible within these plots. A plot with prediction and confidence intervals will be given in section 4.5.3.

Plot 4.11 (a) shows the input size drawn against the runtime. The excerpt covers the runtime for 1 and 8 virtual machines. The data points are colored depending on the amount of $cpus \cdot vms$, which varies from 1 to 64. The used coloring used is also shown in the legend on the right. $ram \cdot vms$ is fixed to 64. All points are also displayed with a standard deviation of $\pm 2 \cdot 20.02\%$ of their value, which is derived from the deviation of all measurements.

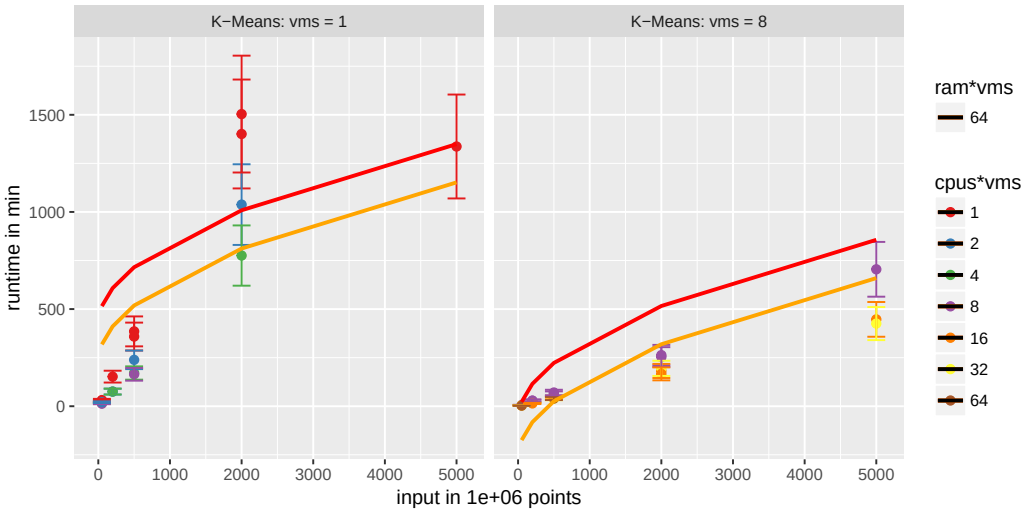
Additionally, two regression curves are plotted, colored according to the amount of $cpus \cdot vms$, in this case the red one shows the fitting for $cpus \cdot vms = 1$ and the orange one for $cpus \cdot vms = 16$. All lines have the same slope in common, but differ in the intercept.

If all variables would be valued in equation (4.9) without the independent in this plot, the resulting regression function according to the left red curve can be written by inserting $vms = 1$, $cpus \cdot vms = 16$ and $ram \cdot vms = 64$ as:

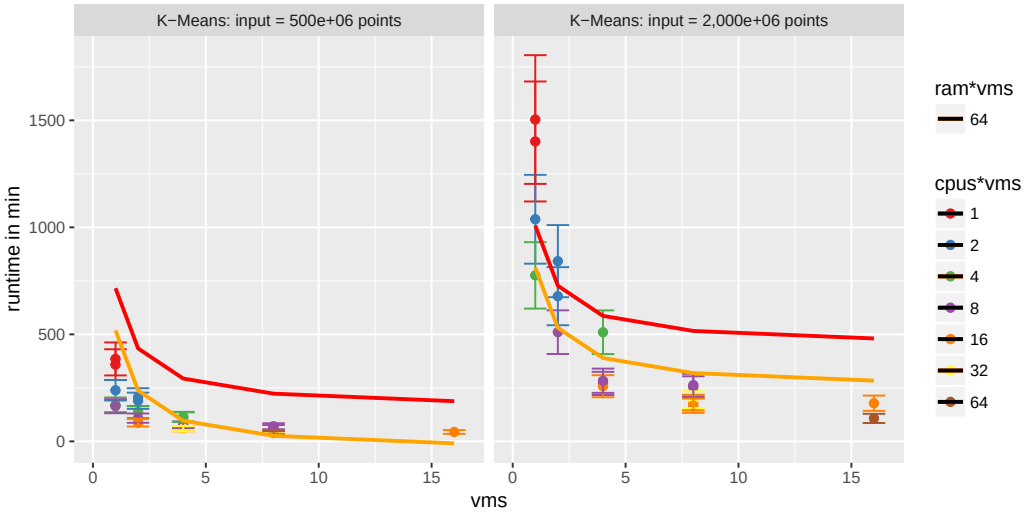
$$\begin{aligned} \mathrm{Y} = t_{runtime} &= \begin{bmatrix} 1 & \frac{1}{1} & \sqrt{input} & \exp\left(\frac{1}{16}\right) & \exp -64 \end{bmatrix} \cdot \begin{bmatrix} -2.79\text{e}7 \\ 3.38\text{e}7 \\ 7.86\text{e}2 \\ 7.15\text{e}6 \\ -1.25\text{e}9 \end{bmatrix} + 0 \\ &= 1.35\text{e}7 + 7.86\text{e}2 \cdot \sqrt{input} \end{aligned}$$

As can be seen, the runtime increases with a gradient input size, given through the transformation with the square root $\sqrt{input}$. The slope is hence defined by the input-coefficient of the regression model and the intercept by the sum of the other dimensions. The negative runtimes, which are logically not possible, occur at a very low input size with a higher product of virtual machines and CPUs per VM. These prediction values are misfits due to a generally more accurate fit on the majority of the dataset, which has an up to a hundred times larger input size. In general, the prediction behavior is still conform with the expectations and the results of the observations.

In plot (b) of figure 4.11 the decreasing runtimes of the observations are plotted dependent on the values of vms. Similar to 4.11 (a), representing regression parameters are defined by $cpus \cdot vms = 1$ for the red and $cpus \cdot vms = 16$ for the orange curve, which differ again only in the intercept, not in the slope and shape.

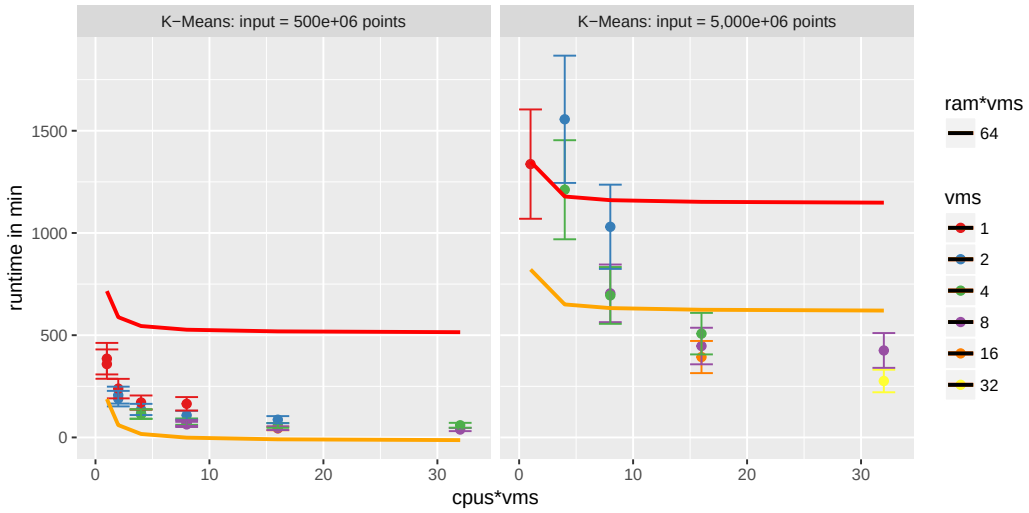


(a) $runtime \sim\ input$, $cpus \cdot vms$ coloured, vms splitted, $ram \cdot vms = 64$, fitted runtimes for $cpus \cdot vms = 1$ in red and $cpus \cdot vms = 16$ in orange regression curves

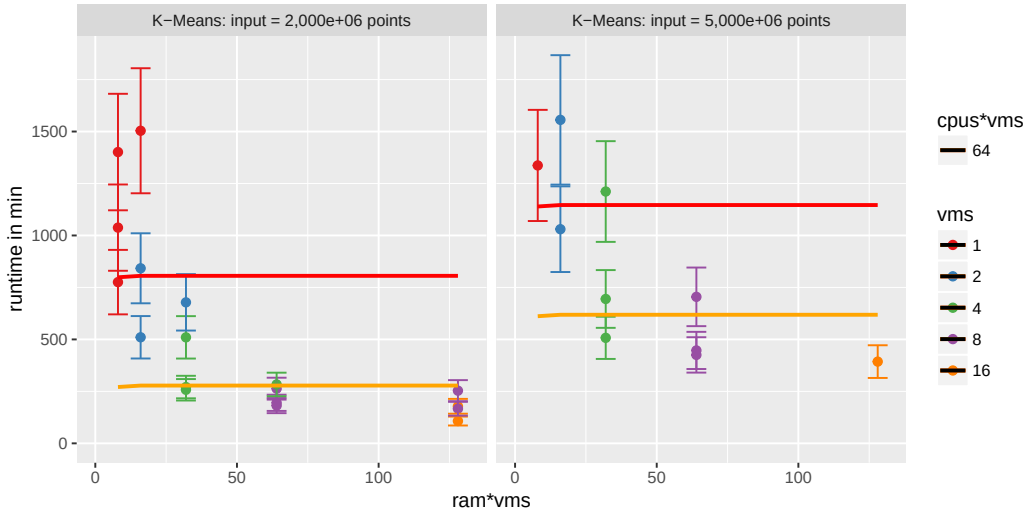


(b) $runtime \sim\ vms$, $cpus \cdot vms$ coloured, $input$ splitted, $ram \cdot vms = 64$, fitted runtimes for $cpus \cdot vms = 1$ in red and $cpus \cdot vms = 16$ in orange regression curves

Figure 4.11: Regression function plots in R against $input$ and vms



(a) $runtime \sim\ cpus \cdot vms$, vms coloured, $input$ splitted, $ram \cdot vms = 64$, fitted runtimes for $vms = 1$ in red and $vms = 16$ in orange regression curves



(b) $runtime \sim\ ram \cdot vms$, vms coloured, $input$ splitted, $cpus \cdot vms = 64$, fitted runtimes for $vms = 1$ in red and $vms = 16$ in orange regression curves

Figure 4.12: Regression function plots in R against $cpus \cdot vms$ and $ram \cdot vms$

With this view the exponential runtime behavior can be seen, defined by the transformation $\exp\left(\frac{1}{cpus \cdot vms}\right)$. For this excerpt the *input* size is set to 50e+07 and 200e+07. The inking indicates that the values of $cpus \cdot vms$ and $rams \cdot vms = 64$ stay the same.

The third dimension $cpus \cdot vms$ is plotted in figure 4.12 (a), with the input size dividing the plots at the values 50e+07 and 500e+07. The observations are colored with the number of vms. The regression functions are defined by $vms = 1$ for the red and $vms = 16$ for the orange one and $rams \cdot vms = 64$ stays still fixed.

The shape of these regression functions is similar to the one in 4.11 (b), but not equal. This is caused by the coefficients of the current dimension, which affect the slope of the exponential curve. The inserted values only shift the curve up and down on the runtimes by changing the intercept.

For plot (b) in figure 4.12 $ram \cdot vms$ is plotted against the runtime, again with the input dividing the data at the values 50e+07 and 500e+07. The observations are colored with the number of vms and the regression functions are again settled for $vms = 1$ for the red and $vms = 16$ for the orange one. This time $cpus \cdot vms = 64$ is the fixed value of the last dimension.

The curves increase rapidly on the smallest values. Unfortunately the small combination of $ram \cdot vms$ was only measured with 50e+07 points. Therefore only in the left plot the significant gradient is visible. The rest of the function converges to 0, due to the transformation $\exp(-ram \cdot vms)$.

This behavior corresponds to the measurements and the expectations, because a very low amount of gigabytes RAM per VM slows down the job computation or even makes it impossible, but from a distinct level it has no influence on the execution.

All plotted regression lines have the distinct estimation of the runtime as a value, but as shown before, this model needs to consider an error, which was disregarded so far. However, it will be investigated in the next section introducing the prediction and confidence intervals.

4.5 Validation of the Regression Model

To instantiate the developed methodology in this chapter mainly two preconditions were assumed. On the on hand the normal distribution of the measurements was postulated. It is statistically not possible to classically prove any underlying distribution entirely, but with the *Shaprio-Wilk test statistic* a null hypothesis, which could rather deny a normal distribution, will be presented on an excerpt of the measurements in subsection 4.5.1.

On the other hand, to initially apply a Linear Regression, the linear independence of the contribution dimensions was assumed. To substantiate this assumption any statistical correlation can be calculated, which allows a statistical interference if the used dimensions do not correlate completely. This correlation is shown in subsection 4.5.2.

Furthermore, the so far plotted regression curves did not include any error tolerance. The Linear Regression values this uncertainty by adding the prediction and confidence intervals, which will be introduced in subsection 4.5.3.

An evaluation of the model will be given at the end of this section.

4.5.1 Normal Distribution of Measurements

Testing for assumptions of distributions, especially for normality, has been an important area of statistical research. Shapiro and Wilk presented a statistical procedure for the hypothesis of a normal distribution of the data set in their paper [SW65].

The problem thereby is, that a classical proof, as used in other problems, is usually not possible, because the underlying data never follows the normal distribution entirely. So it is always an estimation if the presumption of the distribution can be made or not. In general, it is easier to reject a distribution than to prove it. However, as stated in the paper, the use of statistical techniques, like the here used Linear Regression, is in many cases more robust than the underlying assumption and hence can also be applied on data sets where the normal distribution is not completely assured.

The null-hypothesis presented in [SW65] defines the observations as normally distributed. The presented test statistic is given by

$$\mathrm{W} = \frac{\left(\sum_{i=1}^{n} a_i x_{(i)}\right)^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

where x are the observation values, n the sample size and a defines expected values from a standard normal distribution. The null hypothesis is rejected if the derived *p-value* (compare section 4.4.2) is less or equal 0.1. In R the value can be computed and the result is given in listing 4.5.

Listing 4.5: Shapiro-Wilk normality test in R

```
> shapiro.test(measurements)

        Shapiro-Wilk normality test

data:  measurements
W = 0.96367, p-value = 0.7827
```

The used example of the measurements consists of 14 measurements with the configuration $vms = 4$, $cpus = 1$, $ram = 4$ and $input = 5e{+}07$. The Shapiro-Wilk normality test computes a `p-value` of 0.7827, which is greater than 0.1. Hence the null hypothesis is not rejected and therewith a normal distribution can be assumed.

4.5.2 Dimension's Correlation and Independence

For the Linear Regression all measurements were also assumed to be independently distributed. It is an initial condition, to apply such a regression model. For any other case, if they were not normally distributed, another likelihood function and probably also another regression type would have been necessary.

To investigate this, the dimension's correlation is analyzed. It is a statistical measurement how strongly two variables have a linear relationship with each other. The measure of this correlation is called the coefficient of correlation and can be calculated in different ways. The most usual measure is the Pearson coefficient, which is described in [HJ15]. Therefore, the empirical covariance COV between a pair of data (x_i, y_i) is needed, which is

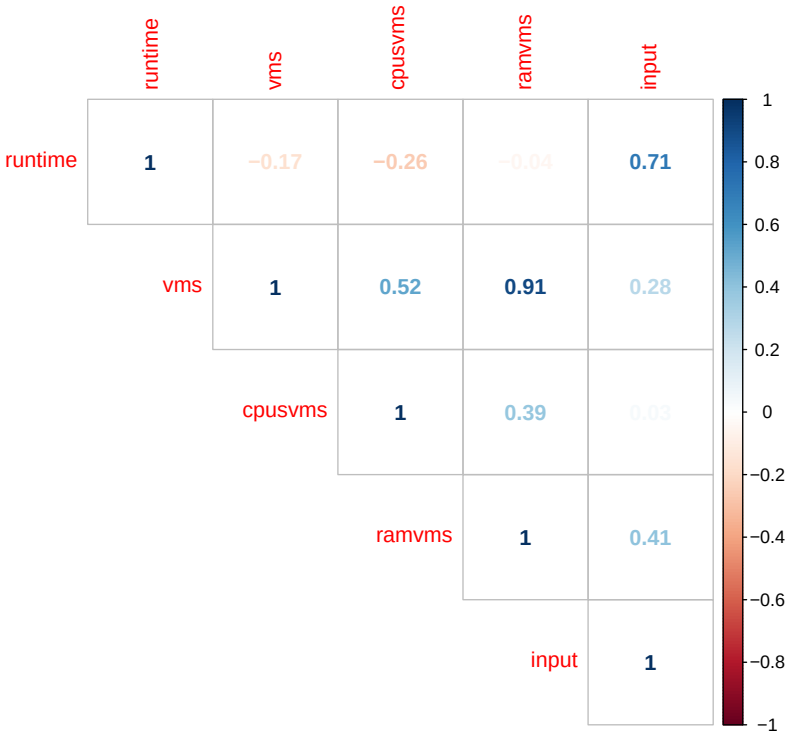


Figure 4.13: Correlation plot of regression dimension from R

$$\mathrm{COV}(x,y) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

where n is the size of the observations. With this definition, the correlation COR is given by

$$\mathrm{COR}(x,y) = \frac{\mathrm{COV}(x,y)}{\sigma_x \sigma_y}$$

where σ_i is the correspondent estimate of the standard deviation for the observations. The correlation has to be between -1 to 1 and for the case equal to -1 or 1 exists a direct linear relationship and thus the regression would not be valid. The closer the correlation is to these two values, the stronger is the dependence. 0 would indicate no correlation. In R it is possible to gain these values with the function $cor(data)$. The values are shown in table 4.8, and in a plot in figure 4.13, where the transparency of the printed values indicates no correlation, but the full blue and red colors a direct one.

As it can be seen, the three independent dimensions vms, $cpu \cdot vms$ and $input$ have no or rather only less correlation. The combined dimension $ram \cdot vms$ do not correlate completely with the underlying dimensions but more than the others. However, because no correlation is 1 or -1, the regression is valid and no direct dependence could be investigated. The dependent variable $runtime$ is also shown, which mostly depends on the input size, which already could be seen in the result interpretation before.

	runtime	*vms*	*cpus* · *vms*	*ram* · *vms*	*input*
runtime	1.000	-0.169	-0.259	-0.042	0.712
vms	-0.169	1.000	0.524	0.909	0.277
cpus · *vms*	-0.259	0.5240	1.000	0.389	0.034
ram · *vms*	-0.042	0.909	0.389	1.000	0.406
input	0.712	0.277	0.034	0.406	1.000

Table 4.8: Correlation values of regression dimensions.

4.5.3 Confidence and Prediction Interval

The estimated value for any prediction at a given point X is derived from the calculation in the regression model. The definition in equation (4.9) needs to be extended by a standard error to create a prediction interval, as stated in [HJ15]. This is very important, because a prediction by itself does not disclose how precise the prediction may be expected. To calculate such an error term, some preliminary definitions are necessary.

The variance of the estimators is calculated with the covariance matrix of the model. The covariance COV of two random variables is a measure of dependences as stated in section 4.5.2. In this usage, it is defined for the estimates of the regression as:

$$\mathrm{COV}(\beta_i, \beta_j) = \mathrm{E}\left[(\beta_i - \hat{\beta}_i)(\beta_j - \hat{\beta}_j)\right]$$

The covariance between the same random variables, here the regression coefficients, is also called the variance $\mathrm{VAR}(\beta) = \mathrm{COV}(\beta, \beta) = {\sigma_\beta}^2$, which is also the same as the squared standard deviation.

With that definition the *Variance-Covariance Matrix*, the matrix of the estimated covariances between the parameter estimates in the linear regression model, can be defined as:

$$\mathrm{V} = \begin{bmatrix} \mathrm{COV}(\beta_1, \beta_1) & \cdots & \mathrm{COV}(\beta_1, \beta_\mathrm{n}) \\ \vdots & \ddots & \vdots \\ \mathrm{COV}(\beta_\mathrm{n}, \beta_1) & \cdots & \mathrm{COV}(\beta_\mathrm{n}, \beta_\mathrm{n}) \end{bmatrix}.$$

The variance of the complete prediction is defined as the product of the prediction vector times the Variance-Covariance Matrix times the transposed prediction vector:

$$\sigma^2_{\mathrm{pred}} = \mathrm{X}_{pred} \cdot \mathrm{V} \cdot \mathrm{X}^T_{pred}$$

Confidence Interval

On account of these previous definitions the confidence interval can be calculated. Therefore, with a confidence level of 95%, the true value of the estimated coefficient is located within this interval, as defined in [HJ15]. This two-sided interval takes the form of an estimate plus and minus a t-quantile (usually $1 - 0.95$) times the standard error:

$$\mathrm{I}_{conf} = \mathrm{t}_{\mathrm{pred}} \pm (\mathrm{T}_{n-p,1-0.95} \cdot \sqrt{\sigma_{pred}})$$

where $T_{n-p,1-0.95}$ is the is the looked quantile of the t-distribution with given n degrees of freedom (the amount of observations) and p independent variables. Its value can be found in the respective literature[9].

Prediction Interval

To specify the interval in which the true value of the prediction is located with a distinct confidence (usually also 95%), the prediction interval needs to be calculated. It is wider than the confidence interval, because it also includes an uncertainty of noise of σ_{pred}, which is estimated with the Pearson correlation coefficient described in [HJ15] as

$$\sigma'_{pred} = \frac{\text{RSS}}{d}$$

where d are the degrees of freedom of the model. Therefore, if the regression model has a huge residual standard error (compare equation (4.8)) and only a finite number of observations, the possible range of the true prediction value gets noticeably wider. The resulting prediction interval is hence given by

$$I_{pred} = t_{\text{pred}} \pm (T_{n-p,1-0.95} \cdot \sqrt{\sigma_{pred} + \sigma'_{pred}}).$$

As shown, the confidence and prediction intervals are very meaningful instruments when evaluating the gained regression model. To give a subsequent example, the use case K-Means will be investigated. R also gives the possibility to determine the independent variables first. The R output of the command `confint(model, level=0.95)` returns their confidence level, shown in table 4.9.

	2.5%	estimation	97.5%
$(Intercept)$	-3.56e+07	-2.79e+07	-2.01e+07
$\sqrt{input}$	6.73e+02	7.86e+02	8.99e+02
$\exp\left(\frac{1}{cpus \cdot vms}\right)$	1.09e+05	7.15e+06	1.42e+07
$\frac{1}{vms}$	2.11e+07	3.38e+07	4.64e+07
$\exp(-ram \cdot vms)$	-1.77e+09	-1.25e+09	-7.19e-08

Table 4.9: 95% confidence interval of the estimated coefficients.

Thus, for one unit increase of one variable (transformed dimension), the mean of the dependent dimension ($runtime$) will increase by between the 2.5%- and the 97.5%-confident value units of the dimension with a 95% confidence.

Hence, for one unit increase in $\sqrt{input}$, for example if the input increases from 25 to 36: $\sqrt{25} \rightarrow \sqrt{36} : 5 \rightarrow 6$, with a confidence of 95% the mean runtime will increase by between 6.73e+02 and 8.99e+02 units (milliseconds in the use case). The same is valid for the other dimensions with the given intervals.

[9] `https://stat.ethz.ch/R-manual/R-devel/library/stats/html/TDist.html`

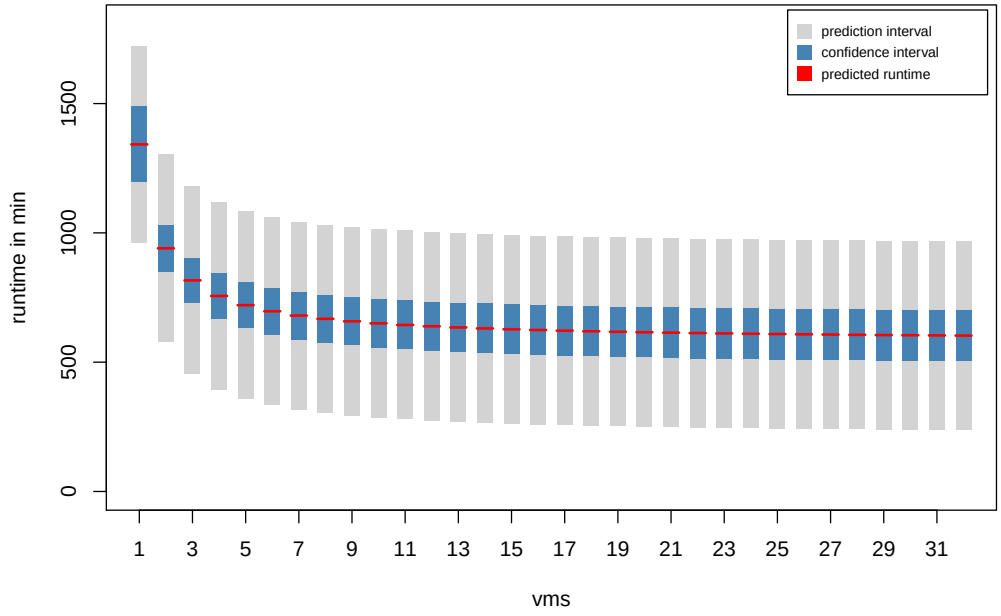


Figure 4.14: Plot of the Confidence and Prediction Intervals depending on the number of *vms*, with set values for $cpus = 1$, $ram = 8$ and $input = 5e{+}09$ points

These intervals can only be specified in combination with a configuration of the independent variables to predict. Table 4.10 shows exemplary confidence and prediction intervals for the response value runtime on given configurations.

				prediction interval				
					confidence interval			
						predicted		
vms	cpus	ram	input			runtime		
2	4	8	1e+09	9.37	302.93	**366.65**	430.37	723.92
1	1	8	5e+09	961.41	1,195.85	**1,342.21**	1,488.56	1,723.00
21	1	8	5e+09	249.50	517.43	**614.10**	710.77	978.69

Table 4.10: Exemplary configurations and predictions from the regression model in minutes

In figure 4.14 the regression function with an x-value of *vms* is plotted against the *fitted runtime* values. The light gray shaded area covers the 95% prediction interval, wherein the blue shaded area defines the confidence interval. In other words, the expected real runtime of the prediction is included into the light gray area with a confidence of 95%, as calculated from the regression model in R.

4.5.4 Evaluation of the Instantiation

The underlying use case caused a very large prediction interval as shown in the last section. From these values, the predicted true runtime value with a confidence of 95% can be expected within in a runtime defined by

$$9.37min \leq t_{runtime} \leq 723.92min.$$

That is an estimation of about 714.55 minutes, respectively 11.91 hours. This very unspecific estimation is caused by the uncertainty of the observations and the hence not optimal regression model. The standard deviation of all measurements was at 20.02% in the use case K-Means.

Hence, although the prediction interval is that wide, it is not a reference that all real runtimes are that widely spread. It only states, that every measured runtime of a given configuration that lays in this prediction interval around the predicted runtime needs to be accepted due to the fact of the huge deviation.

Applying this methodology to any applications with measurements taken in an unshared, exclusive cloud environment, should essentially scale down the uncertainty of the prediction and improve the $R^2_{adj.}$ value of the regression model. Within this thesis such measurements could not be achieved, but will be suggested as a future work project in chapter 6.

This instantiation presented the complete process of gathering measurements, determining the relevant dimensions, arranging their transformations within the regression formula and calculating the estimators of the coefficients of the Multiple Linear Regression, and additionally providing all mathematical equations for the calculation.

The resulting regression model in this chapter is only suitable for the defined configurations of the cloud environment and the virtual machines, especially the processing framework Flink and the distributed *LRZ compute cloud*. It is therefore not a general solution for K-Means in a cloud environment and also no generalization of any Flink implementation. It is the product of this instantiation within the defined scope.

The underlying methodology generally described before can be in contrast applied to any cloud infrastructure, with any processing framework on any possible configuration of a virtual machine. The next chapter will continue the evaluation of the methodology and provide the basic principles of concrete predictions within the use case K-Means, but also a second application, Wordcount.

5 Runtime Prediction and Regression Model Evaluation

This chapter presents the validation of the determined regression models by predicting runtime values for different configurations and comparing them to actually measured ones. Arranged according to the instantiation of the methodology in chapter 4, section 5.1 will provide the mathematical definition of any prediction.

Beyond the initial scope of this thesis an additional prediction of an optimal execution configuration based on values of a subset of the problem space is presented in section 5.2. Hence it is possible, for example, to select the resource optimized configuration for a specified runtime and input size.

These prediction values will be used to validate any regression model gained from the presented methodology. The validation process will be explained in section 5.3.

Afterwards runtime predictions of the clustering algorithm K-Means will be calculated in section 5.4. A brief summary of the obtained regression model will be given in section 5.4.1 before predictions and actual measurements are compared in section 5.4.2. The evaluation will conclude with the validation of the regression model.

With Wordcount another well-known application will be investigated in section 5.5 . The complete methodology for predicting the runtime behavior is applied to this application and the findings are presented in section 5.5.1. The predictions based on these results are challenged with corresponding measured runtime values in section 5.5.2, and used to evaluate and validate the regression model afterwards.

This chapter will close with an analysis of the investigated standard deviation of the initial measurements and the differences between the prediction and actual measured runtime values in section 5.6.

5.1 Prediction

The regression model obtained from the methodology, for example instantiated in chapter 4, will be used to predict runtime values of an analyzed application by altering the values of the independent variables. With the calculated estimators of the regression coefficients β^* the response value Y is represented by

$$\mathrm{Y} = \mathrm{X}_{\mathrm{pred}} \cdot \beta^* + \varepsilon$$

where $\mathrm{X}_{\mathrm{pred}}$ is the transposed vector including the variable values on which the runtime should be predicted. ε provides the additional term of any uncertainty described within the confidence and prediction intervals. The vector with which the runtime should be predicted needs to have values inserted for each dimension and 1 for the intercept:

$$\mathrm{X}_{\mathrm{pred}} = \begin{bmatrix} 1 & X_1 & \cdots & X_p \end{bmatrix}.$$

The values are inserted by applying the dimension's transformations. With the given use case in chapter 4 the transposed prediction vector would have the shape

$$\mathrm{X_{pred}} = \left[1 \quad \frac{1}{vms_{pred}} \quad \sqrt{input_{pred}} \quad \exp\left(\frac{1}{cpus_{pred} \cdot vms_{pred}}\right) \quad \exp\left(-ram_{pred} \cdot vms_{pred}\right)\right].$$

for which in each dimension's transformation any value could be inserted. The prediction scope should usually be limited to a certain range of values that are technically achievable in all dimensions.

Runtime Prediction

The estimated runtime prediction $\mathrm{t_{pred}}$ is hence calculated from the product of the prediction and the coefficient vectors:

$$\mathrm{Y} = \mathrm{t_{pred}} = \mathrm{X_{pred}} \cdot \beta^* = \begin{bmatrix}1 & X_1 & \cdots & X_p\end{bmatrix} \cdot \begin{bmatrix}\beta_0 \\ \beta_1 \\ \vdots \\ \beta_\mathrm{p}\end{bmatrix} + \varepsilon \tag{5.1}$$

With the given equation the methodology offers the possibility to predict the runtime of a given configuration within a certain prediction interval. These runtime predictions enable a validation of any regression model. This will be presented in section 5.3.

5.2 Configuration Prediction

Besides this runtime prediction, the estimators of the regression model can also be used to predict a suitable configuration of a subset of the problem space. Therefore the remaining subset should be specified including the response value.

A naive approach calculates the response value for each possible combination of the configuration subset and compares it to the provided values. The configuration with the smallest difference in the response dimension is the most suitable.

As an example of the use case K-Means, a target runtime of 7 hours and an input size of 20e+07 points are defined. The challenge is to determine the optimal configuration of the remaining dimensions. For all these dimensions the runtime values of all cross combinations are calculated with the defined constant input size. The resulting predictions from the regression model are compared to the target runtime and, for example, listed in a table. The optimal configuration can be found where the difference between the predicted and the specified runtime value is the smallest. For the unset dimension also limitations can also be specified, which corresponds to real scenarios where the maximum amount of each dimension value is usually limited.

The solution of the given use case is indicated in table 5.1, while the dimension values were limited with $vms \leq 64$, $cpus \leq 8$ and $ram \leq 32$.

An enhanced algorithm could also define weights for specific dimensions, focus cost or resource optimization, and will be an aspect of the later presented continuing work after this thesis.

vms	cpus	ram	input	Δt in s
...				
35	7	1	20e+07	2.490
35	8	2	20e+07	1.171
35	8	3	20e+07	1.171
...				

→

vms	cpus	ram
35	8	2

Table 5.1: Select optimal configuration within given runtime and input size

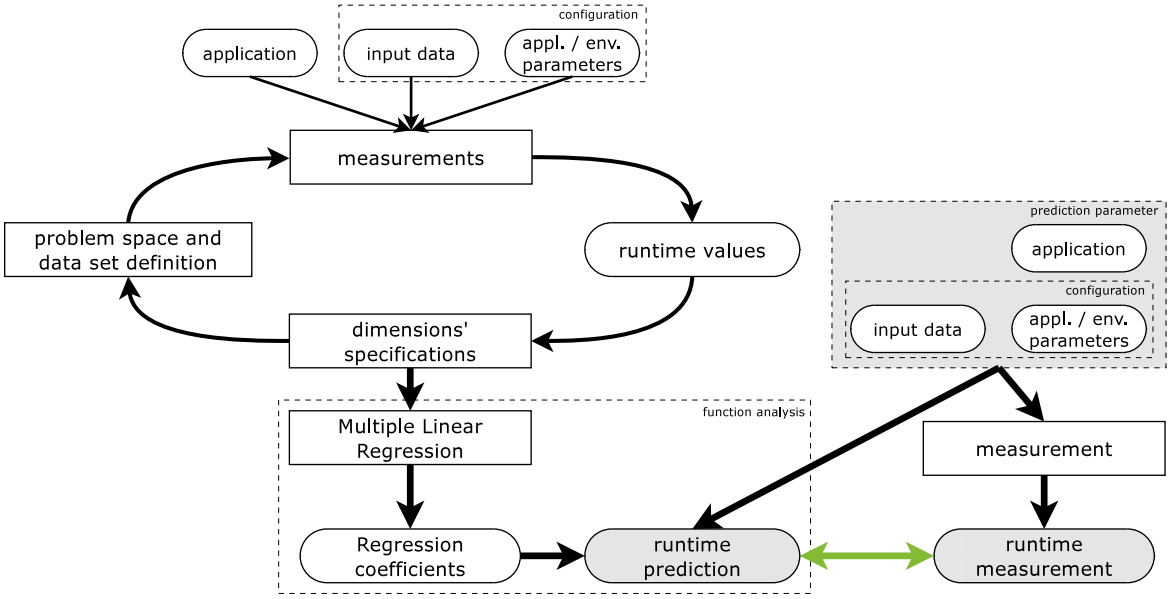


Figure 5.1: Validation process of instantiated Methodology by runtime comparison

5.3 Validation of a Regression Model

The methodology presented in chapter 3 is extended to enable a validation possibility for the determined regression model. Therefore, as shown in figure 5.1, the same application but with another configuration of its input and the virtual machines will be applied on the regression model to predict a runtime value. The same configuration will be used to measure the runtime value in the execution environment.

Both results are compared afterwards, and if the actual measured runtime value is included within the prediction interval of the predicted runtime, the regression model can be validated. The number of single predictions and corresponding measurements should be distributed on the complete range of the underlying problem space. An extrapolation of these values is possible and can also validate the model, but needs to be limited within the general scope of the regression model.

points	centroids	iterations	file size
8e+07	100	10	0.944 GB
40e+07	100	10	4.720 GB
100e+07	100	10	11.801 GB
300e+07	100	10	35.801 GB
800e+07	100	10	94.408 GB

Table 5.2: K-Means application configurations for validation measurements

5.4 Use case K-Means

The application K-Means was introduced in section 4.2.1 as an iterative clustering algorithm. Within the instantiation of the methodology the dimension values and the outcome of the regression were calculated. Section 5.4.1 will give a compact summary of these findings before the prediction is applied on the application in section 5.4.2.

5.4.1 Application of the Methodology

The measured data set initially contained 450 single measurements in the altered dimensions *vms*, *cpus*, *ram* and *input* size, having a standard deviation of 20.02%. From them, together with the added pair-wise combined dimensions, each corresponding transformation was determined. The statistically significant dimensions were identified by the AIC.

On them, brought together to the regression formula, a Multiple Linear Regression was applied. No Outlier and no Leverage point was removed. The calculated estimators of the regression coefficients as well as the significant dimensions with their transformation are given in the following equation:

$$\mathrm{Y} = t_{runtime} = \begin{bmatrix} 1 & \frac{1}{vms} & \sqrt{input} & \exp\left(\frac{1}{cpus \cdot vms}\right) & \exp\left(-ram \cdot vms\right) \end{bmatrix} \times \begin{bmatrix} -2.79\text{e}7 \\ 3.38\text{e}7 \\ 7.86\text{e}2 \\ 7.15\text{e}6 \\ -1.25\text{e}9 \end{bmatrix} + \varepsilon$$

With this solution, the runtime can be predicted for new values of the transposed vector $\mathrm{X}_{\mathrm{pred}}$ as shown in equation (5.1).

5.4.2 Runtime Predictions

For several new configurations (values of $\mathrm{X}_{\mathrm{pred}}$), the runtime prediction as well as the confidence and prediction intervals are calculated with R. Besides not used hardware combinations also new input files were created, following the same form as the measured ones in the use case before and listed in table 5.2

According to figure 5.1 the validation process was executed on ten different configurations. Table 5.3 shows the selected configurations and their predicted values.

As stated before, the negative values derive from the large prediction interval expanding in positive and negative values. It can also be seen, that the prediction intervals, which should include the true runtime with a confidence of 95%, are very large. The average interval range

vms	cpus	ram	input	prediction interval	confidence interval	predicted runtime	measured runtime	confidence interval	prediction interval
2	1	4	8e+7	-233.58	58.95	**123.92**	**32.55**	188.89	481.42
2	8	8	8e+7	-298.76	-16.51	**61.29**	**10.34**	139.10	421.35
1	1	8	4e+8	302.80	547.46	**677.70**	**248.93**	807.95	1,052.60
2	2	8	4e+8	-123.71	176.02	**232.31**	**117.97**	288.61	588.34
4	8	8	4e+8	-293.53	11.85	**61.51**	**46.76**	111.17	416.55
2	4	8	1e+9	9.37	302.92	**366.65**	**193.30**	430.37	723.92
15	2	8	1e+9	-245.57	52.09	**110.85**	**57.92**	169.61	467.27
11	4	16	3e+9	67.77	354.86	**426.55**	**150.14**	498.23	785.33
14	2	4	8e+9	498.55	747.01	**871.49**	**421.70**	995.97	1,244.42
21	4	8	8e+9	481.84	729.50	**855.19**	**289.30**	980.85	1,228.51

Table 5.3: Predicted and measured runtime as well as confidence and prediction interval values given in minutes for the use case K-Means

has a size of 718 minutes, which are about 12 hours. This means every runtime prediction with a 95%-confidence follows the equation

$$t_{95\%} \sim \ t_{pred} \pm 6h.$$

Statistically, all measured runtimes within this interval would be sufficient to validate the regression model. For any user such a large prediction interval is in many cases insufficient, because more accurate predictions are desired. Nevertheless, by adding the standard deviation of measurements to all actually measured runtimes, all but one of the configurations measured can validate the regression model.

To visualize these values, all measurements are additionally plotted in figure 5.2 against the runtime in minutes. The configurations are labeled in the values of the x-axis (v: vms, c: cpus, r: ram, i: input,) with an additional consecutive number. For each the prediction interval is plotted in gray and the confidence interval in blue. In the center of each interval the predicted runtime value is marked in red. The mandatory added standard deviation of each measurement is illustrated with the error bars up and down the measurement.

The actually measured runtime with its deviation bars, $\pm 20,02\% \cdot runtime$, are plotted in black. All explanations can also be seen in the legend on the upper left.

As stated and graphically proved, all actual runtimes but the last could be measured within the prediction interval. The estimations tend to predict too long runtime values. It seems that the calculated regression coefficients cannot represent the scaling behavior correctly.

The measurements (3) to (5) with an input size of 40e+07 points have an increasing number of CPUs and hence Flink task managers from $1 \rightarrow 4 \rightarrow 32$, which should reduce the runtime significantly. This behavior can be seen in both, in the predictions as well as in the actual measurements. At measurement (5) both values are close together and can be stated

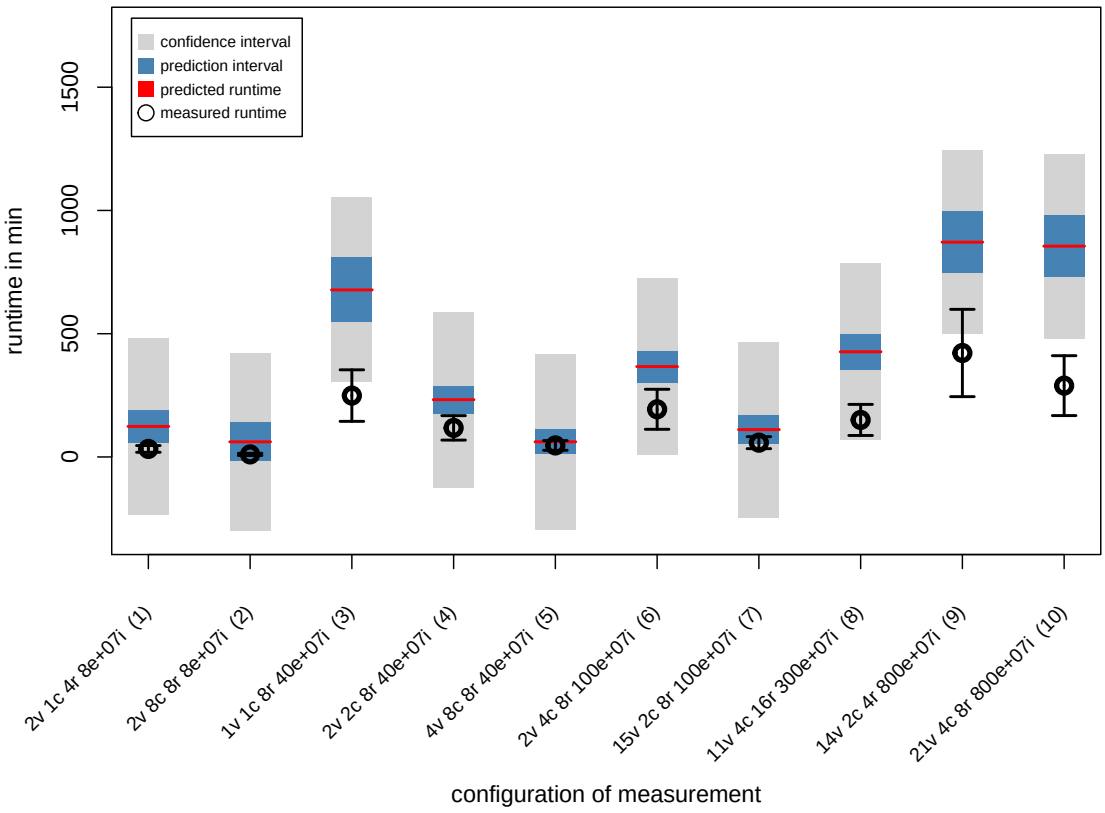


Figure 5.2: Plot of predicted and measured runtimes as well as confidence and prediction intervals for the use case K-Means

overlapping, taking the standard deviation into account. However, for smaller computational power, with less CPUs per VM, the runtime drifts apart.

The reason for this behavior, smaller difference of predicted and measured runtimes with more involved virtual machines, might be found by investigation of the application and the initially measured runtime values. If the number of virtual machines increases, the load on each VM is reduced, as well as the execution time. K-Means has an intensive computational phase while calculating the distance from each point to each centroid. Because this computation time is reduced, the complete job execution gets more influenced by peaks in the load on the underlying hardware nodes. Therefore, it might be reasonable that the predicted runtimes do not cover the actually measured ones, because the initial observations influenced the regression model with misleading and high deviated runtime values.

However, this behavior cannot be finally analyzed, because no more measurements within this thesis were possible. This was influenced by time constraints caused by the Flink environment, which let all but two measurements with large input files fail due to untraceable errors. So only two measurements (9) and (10) for the highest input size at 8e+09 points (~100GB) were successful.

These observations indicate a similar behavior as the three observations mentioned before. The actual runtimes are much lower than the predicted ones. The contributing 28 respectively 84 task managers should not be out of the scope of the regression model in general, which was initiated with a maximum of 32 VMs. Yet as stated before, only one such high observation could be done before instantiating the methodology. It is expected, if more measurements at a higher input size and higher amount of combined task managers would be initially added to the regression, they would improve the outcome and thereby would also improve the estimation of any new configurations.

A repetition of these missing measurements at a higher input size and other optimizations regarding the measurements in a general view are also summarized in the future work suggestions of this thesis in chapter 6.

Nevertheless, these comparisons of the presented predicted and measured runtime values endorse the conclusion that the determined regression model for the application K-Means according to the disposed methodology can be validated.

5.5 Use case Wordcount

The second application is a kind of "hello world" program of big data frameworks. Its details can also be found in the Apache project documentation[1].

Wordcount computes the amount of appearances of a word in a plain text. Therefore, the algorithm is divided into two steps. In the first step the input text is split into individual words. Afterwards, these words are grouped by the same word and counted. The Flink Java implementation computes a simple word appearance histogram over the text files, which are again only plain text with lines separated by newline characters.

Figure 5.3 shows an example how Wordcount counts words and reduces them to a grouped output.

The basis used to create the input files was downloaded from the Gutenberg project[2]. The

[1] `https://ci.apache.org/projects/flink/flink-docs-release-1.3/api/java/org/apache/flink/examples/java/wordcount/`

[2] `http://www.gutenberg.org/`

"To be, or not to be, that is the Question"[3]	→	(2, to) (2, be) (1, or) (1, not) (1, that) (1, is) (1, the) (1, question)

Figure 5.3: Wordcount example of counting words

entire collection of over 54,000 single books were arranged in new text files. Therefore no artificial letter combinations were created, but real world text files. The concretely used file and text sizes, as there is only one input file to the application, are listed in table 5.4.

words	file size	amount of measurements
177e+06	1,057 GB	78
702e+06	4,188 GB	163
1,645e+06	10,810 GB	154
14,935e+06	100,926 GB	5

Table 5.4: Wordcount input sizes for measurements

In this use case, the input amounts of *words* and *file size* are not equivalent. This is caused by the underlying single texts, respectively books, being different, with different words of different sizes. Although the file size is more impartial, it has no information on the processing amount, as the input is processed by the amount of words and not the file size. The absolute number of words counted in the file is used as a value for the methodology in this thesis.

5.5.1 Application of the Methodology

Data Set

For this use case, the methodology is executed completely again with the data set of the Wordcount measurements. It contains 400 single measurements. Unfortunately, similarly to the use case K-Means presented before, the measurements with the highest input size are underrepresented. From the complete 400 observations, only 5 could be gained with the input size of 100 GB and almost 15 billion words. The other amounts can be found in table 5.4.

All these single measurements were also executed in the *LRZ Compute Cloud*. For the analysis, the average runtimes of the same configurations were computed. Therefore, the final observations had an amount of 84 averaged measurements and a standard deviation of 19.18%, similar to K-Means at a very high level. As in the K-Means case, any Wordcount prediction will also face widespread confidence intervals.

Dimensions' Transformations

According to algorithm 1 on each dimension a simple Linear Regression was computed with the response value *runtime*. Each resulting transformation was derived from the highest

[3]Shakespeare, William: Hamlet. Ed. by Harold Jenkins. London, New York: Methuen, 1982.

$R^2_{adj.}$ value. For the five dimensions these are

$$\begin{aligned} f_{vms} &: x \mapsto \frac{1}{x} \\ f_{cpus} &: x \mapsto \ln x \\ f_{ram} &: x \mapsto \exp x \\ f_{input} &: x \mapsto \sqrt{x} \\ f_{cpus \cdot vms} &: x \mapsto \sqrt{x} \\ f_{ram \cdot vms} &: x \mapsto \exp \frac{1}{x} \\ f_{cpus \cdot ram} &: x \mapsto \exp x \end{aligned}$$

Statistical Significance

If a dimension has an impact on the response variable or not is determined with the AIC and BIC method detailed in section 4.2.6, the resulting statistically significant dimensions are

$$\text{runtime}_{\text{AIC}} = \text{runtime}_{\text{BIC}} \sim \begin{bmatrix} 1 & \frac{1}{vms} & \ln cpus & \sqrt{input} & \sqrt{cpus \cdot vms} \end{bmatrix} \cdot \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \end{bmatrix}$$

The dimension *ram* is not statistically significant for this application with the underlying measurements and therefore removed from the dimension set. The runtime averages need to be computed again grouped from scratch without this dimension.

Afterwards the transformations of the dimensions with the new arithmetic mean runtimes need to be determined again. On this new arrangement the AIC and BIC also need to be applied again. This time, with the altered dimensions, the AIC and BIC differ. Their significant dimensions found with the new transformations are:

$$\text{runtime}_{\text{AIC}} \sim \begin{bmatrix} 1 & \exp\left(\frac{1}{vms}\right) & \exp\left(\frac{1}{cpus}\right) & \sqrt{input} & \sqrt{cpus \cdot vms} \end{bmatrix} \cdot \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \end{bmatrix}$$

$$\text{runtime}_{\text{BIC}} \sim \begin{bmatrix} 1 & \exp\left(\frac{1}{vms}\right) & \sqrt{input} \end{bmatrix} \cdot \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix}$$

Applying the BIC would once more lead to a refinement of the problem space and the data set. However, the methodology follows the AIC and the remaining amount of dimensions stays the same. However, *vms* and *cpus* had to change their transformation to $x \mapsto \exp\left(\frac{1}{x}\right)$. All four dimensions are processed into the following function analysis.

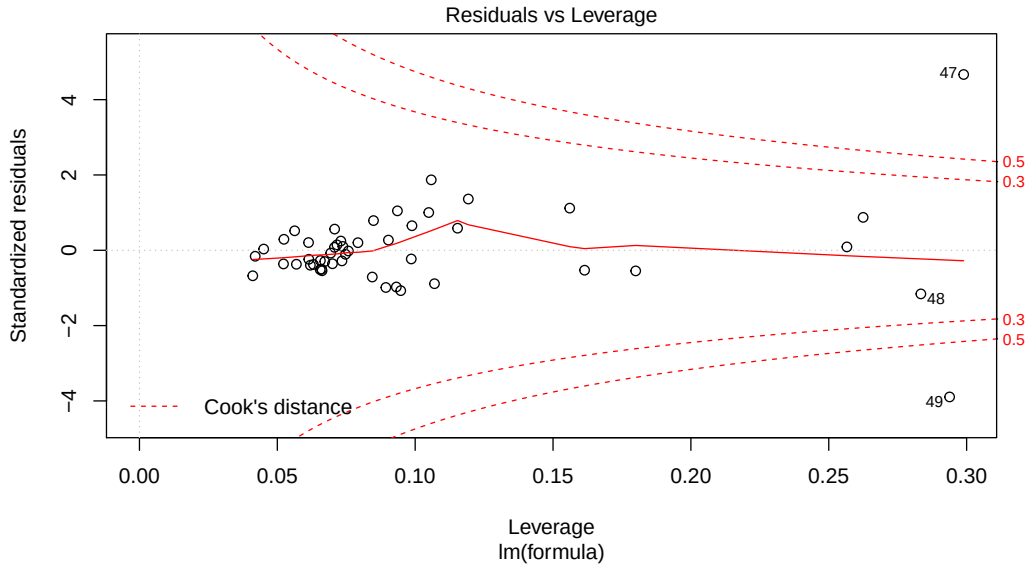


Figure 5.4: Cook's Distance plot for application Wordcount

Function Analysis

The Multiple Linear Regression is applied with the retrieved formula on the averaged data set. Following the methodology all Outlier and Leverage points need to be investigated if any defective measurements should be removed from the data set. The Cook's Distance plot in figure 5.4 shows three obvious Leverage points, respectively Outliers. Their distance value is far above Cook's 0.5 distance.

All of them belong to the observations of the huge 100GB input file. They show no sign of any defective measurement and therefore may not be removed. However, this analysis substantiates the assumption that the number of observations, especially for the highest amount of input words, is much too low.

The red average line is close to an optimal zero horizontal, although the points with a higher distance exist. As stated, this should be a sign for a more or less balanced regression model.

Regression Coefficients

To determine the estimators of the regression coefficients, the MLR is executed, again with the support of R. The resulting output from the computation is shown in listing 5.1.

Listing 5.1: Multiple Linear Regression output from R

```
> summary(model)

Call:
lm(formula = formula, data = data)

Residuals:
      Min         1Q     Median         3Q        Max
-21839259  -3233351   -680517   1891723  26080851

Coefficients:
                     Estimate Std. Error t value Pr(>|t|)
(Intercept)          3.27e+06   9.49e+06    0.34    0.732
exp(I(1/vms))        4.51e+06   2.34e+06    1.93    0.060 .
exp(I(1/cpus))      -5.34e+06   2.12e+06   -2.52    0.015 *
sqrt(input)          4.62e+02   4.08e+01   11.31  1.3e-14 ***
sqrt(I(cpus * vms)) -2.14e+06   9.99e+05   -2.14    0.038 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6670000 on 44 degrees of freedom
Multiple R-squared:  0.762,     Adjusted R-squared:  0.741
F-statistic: 35.3 on 4 and 44 DF,  p-value: 3.31e-13
```

The Linear Regression has an $R^2_{adj.}$ value of 0.741. The relative low amount of observations is penalized. Although the p-value indicates that the regression model explains the observations, the f-statistics is significantly lower than in the K-Means use case (74.1 on 4 and 80 DF). Also, all the single dimensions except for *input* show no strong statistical significance, but enough to improve the model. As Cook's distance plot these regression values also indicate a suboptimal small data set.

5.5.2 Runtime Predictions

Akin to the application K-Means, predicted runtimes are one again compared to actually measured ones, aiming to validate the evolved regression model according to figure 5.1. Also for Wordcount new input files with real world texts were created. The specifications can be found in the following table 5.5:

words	file size
1.352e+09	8,163 GB
5.181e+09	34,880 GB
18.634e+09	124,086 GB

Table 5.5: Wordcount input sizes for validation measurements

The investigated ten configurations are listed in table 5.6, also showing the predicted and actual runtime values as well as the prediction and confidence intervals computed from the regression model in R.

Again, the configurations of the 10 measurements are labeled in the values along the x-axis (v: vms, c: cpus, r: ram, i: input,) with an additional consecutive number. For each the

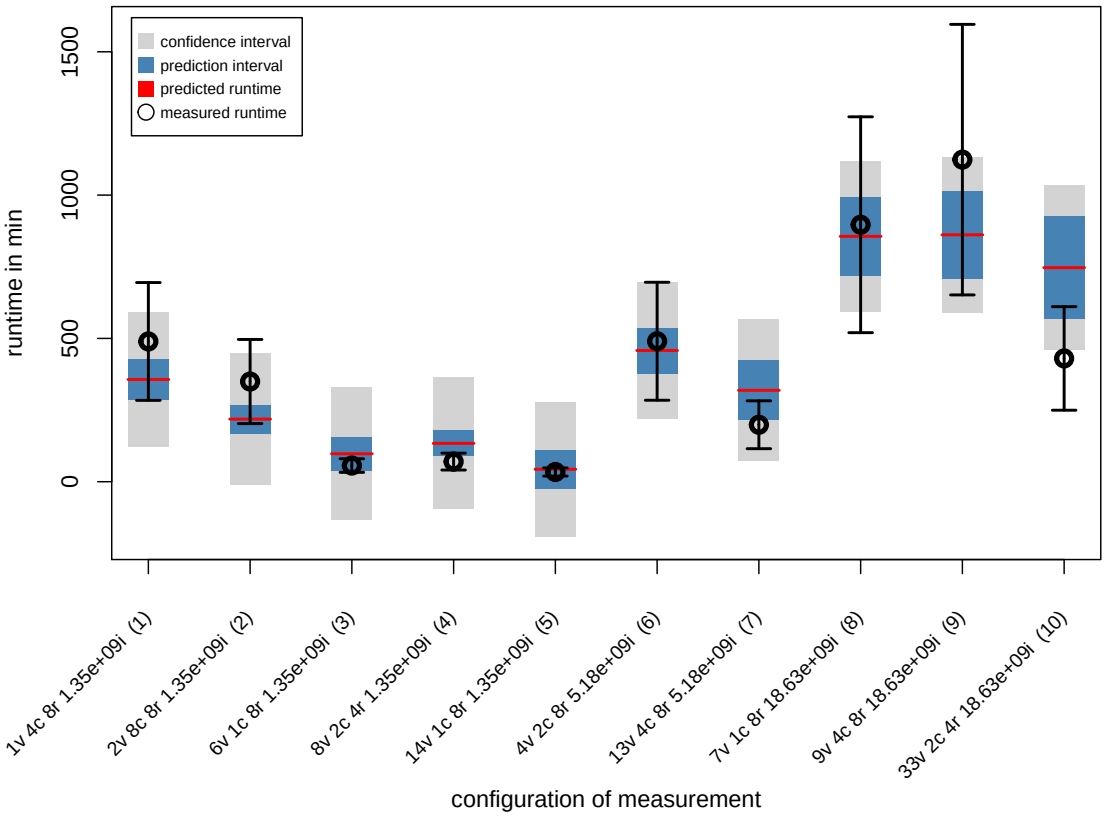


Figure 5.5: Plot of predicted and measured runtimes as well as confidence and prediction intervals for the use case Wordcount

vms	cpus	input		prediction interval				
				confidence interval				
					predicted runtime	measured runtime		
1	4	1.35e+09	121.48	285.46	**356.68**	**489.39**	427.90	591.90
2	8	1.35e+09	-11.12	168.85	**218.45**	**349.57**	268.00	448.00
6	1	1.35e+09	-133.98	39.66	**97.53**	**56.39**	155.40	329.00
8	2	1.35e+09	-94.76	89.06	**133.82**	**70.28**	178.60	362.40
14	1	1.35e+09	-190.55	-23.74	**43.46**	**33.91**	110.70	277.50
4	2	5.18e+09	219.94	378.18	**457.83**	**490.08**	537.50	695.70
13	4	5.18e+09	71.97	215.31	**318.91**	**198.73**	422.50	565.80
7	1	1.86e+10	593.13	718.63	**856.06**	**896.64**	993.50	1,119.00
9	4	1.86e+10	589.98	708.28	**861.50**	**1,123.73**	1,014.70	1,133.00
33	2	1.86e+10	460.05	567.89	**746.93**	**430.15**	926.00	1033.8

Table 5.6: Predicted and measured runtime as well as confidence and prediction interval values given in minutes for the use case Wordcount

prediction interval is plotted in gray and the confidence interval in blue and the predicted runtime value is marked with a red line in the middle of the intervals. The mandatory added standard deviation of the measurement is shown with the error bars up and down the measurement.

In contrast to the results of K-Means, all validation measurements can be included in each corresponding prediction interval and actually also in each corresponding confidence interval.

Measurements (1) to (5) refer to an input size of 1.35e+09 words. The expected scaling behavior with a decreasing runtime by increasing the computational power can be observed. Remarkable regarding measurement (4) is the higher predicted and higher measured runtime value. Although its computational power includes more Flink task managers ($8 \cdot 2 = 16$) than measurement (5) ($14 \cdot 1 = 14$) its runtime is higher. This depends on their composition. While in the first configuration each VM is assigned two CPUs and therefore two task managers, measurement (5) includes only VMs with one task manager on each. As the initial measurements showed, the scaling with more CPUs has less influence on the runtime than scaling up the number of VMs.

For the highest input size of 18.63e+09 words, measurements (8) to (10) show a similar behavior as K-Means. The involved Flink task managers increase from $7 \rightarrow 36 \rightarrow 66$. Normally a gradient decreasing runtime would be expected, but the regression predicts a higher runtime for the 36 task managers than for the 7. The reason is identical to the measurements (4) and (5), where the absolute parallelism doesn't influence the runtime most. It is more dependent on the combination.

The allocated 4 CPUs on each of the 9 VMs at measurement (9) have such an increasing influence on the runtime, that the overall runtime is higher than with only 7 VMs. This behavior could be proved by the measurement and results from the approach of increasing

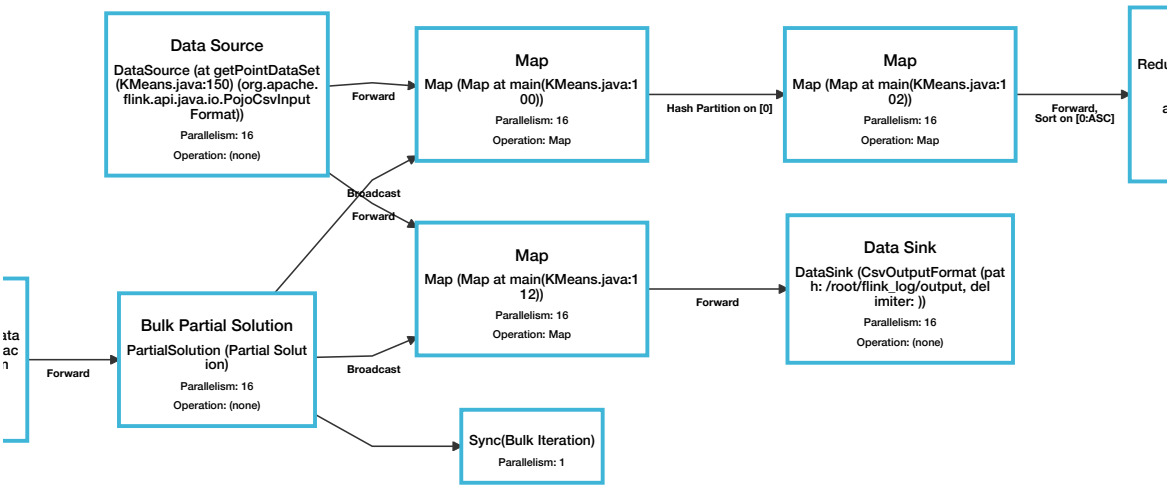


Figure 5.6: Excerpt of a Flink dataflow graph for a K-Means execution

the computational power per vm by adding more task managers and in the same amount also CPUs to the VMs. However because all job executions with more than one CPU per VM are so influenceable by any external load on the hardware node, the initial measurements as well as this validating measurement show an increasing instead of a decreasing runtime behavior.

Although such a behavior is not intended by any user, the taken measurements and their compared predicted values also allow validation of the regression model of Wordcount presented here. The thus twice instantiated methodology could hence be validated for both applications.

5.6 Accuracy of Measurements and Predictions

As presented before, the predicted runtime values for K-Means and Wordcount are within the scope of the statistical uncertainty of the regression models. Their behavior is similar and could represent the scaling runtime values.

However, both models show differences in the accuracy of the prediction. While K-Means' model predictions have an averaged difference between predicted and measured runtime value (runtime delta) of 174%, Wordcount's regression model can predict the runtime value with an averaged delta from the actual runtime value of only 38%. The difference between these two applications might refer to their actual computation phases.

Generally, such phases are divided in different dependencies on hardware or other configurations of the virtual machines. Such phases of applications can be CPU-bound (high CPU-usage), memory-bound (high RAM-usage), disk-bound (high disk-usage) or network-bound (high network traffic).

Figure 5.6 shows the dataflow graph of the Flink job execution of K-Means. This graph is the execution path for the Flink cluster and is computed on the basis of the source code during the job submission by the Flink Client. The single implementation phases, illustrated within the blue boxes, have an explicit operation and a dedicated parallelism, depending on

many task managers are involved. For example the processing of the *Data Source* in the upper left box, which is done by 16 task managers, located on one or more VMs, is disk or network-bound, depending on the locality of the data.

Each relevant operation can be assigned to one of two phases, a *map* or *reduce* phase, which refers the big data processing model *mapreduce* adapted for Flink and described in [DG08]. The map phase describes any, also very complex computation on the data. It can be imagined as a mapping from one initial value to a transformed one or a result derived from it. Therefore, it tends to be CPU- and probably disk-bound, instead of the reduce phase being I/O-bound, e.g. network-bound for the case the task managers are not on the same node. In the reduce phase a counting or summary operation is executed, for example building the sum of all counted words in the Wordcount application. Hence these phases of the application might show other performance behaviors while increasing different vm resources.

The upper row of figure 5.6 defines one iteration, where the input file is split and processed by the task managers. The single steps are divided into map phases, in which the implementation parts within the computation are done. Here, the worker node needs to compute the distance to each centroid for each assigned point. The last step of any iteration is a reduce phase, where the results are sorted, grouped and distributed.

The initial measurements for K-Means showed an overall standard deviation of 20.02%. Remarkable is the fact that the deviation decreases with an increasing input size, at K-Means but also at Wordcount. Table 5.7 lists the standard deviation of the initial measurements of K-Means (a) and Wordcount (b) depending on the input size. As it can be seen, the standard deviation in both cases has its highest values with the smallest input size and averaged runtime.

input in points	av.runtime in min	stand. deviation
5e+07	11.97	26.55%
20e+07	50.070	18.28%
50e+07	130.56	15.64%
200e+07	510.27	18.07%
500e+07	794.86	17.90%

(a) K-Means

input in words	av.runtime in min	stand. deviation
18e+07	20.62	22.60%
70e+07	91.81	21.62%
165e+07	256.56	17.15%
1,493e+07	625.09	10.33%

(b) Wordcount

Table 5.7: Standard deviation of initial measurements dependent on input size

This behavior, whereby the measurements are less deviated, the longer the duration of a job execution is, should be investigated intensely. From this point of view, one likely reason might be that the performance of the allocated cloud resources gets balanced over the time, still with a huge influence on the overall performance, but with continuing less peak performances in positive or negative direction.

What can be surely stated is, that the long-term job executions are more reliable than the short runs. This can also be observed in the comparison of the predicted and measured runtime values of the sections before. Figure 5.7 shows the difference between the runtime deltas of both applications in percentages of the actually measured runtime values.

K-Means has a very high averaged delta runtime of 174%, but it becomes reduced the higher the input size grows. The same plot for Wordcount shows a similar behavior, but in a

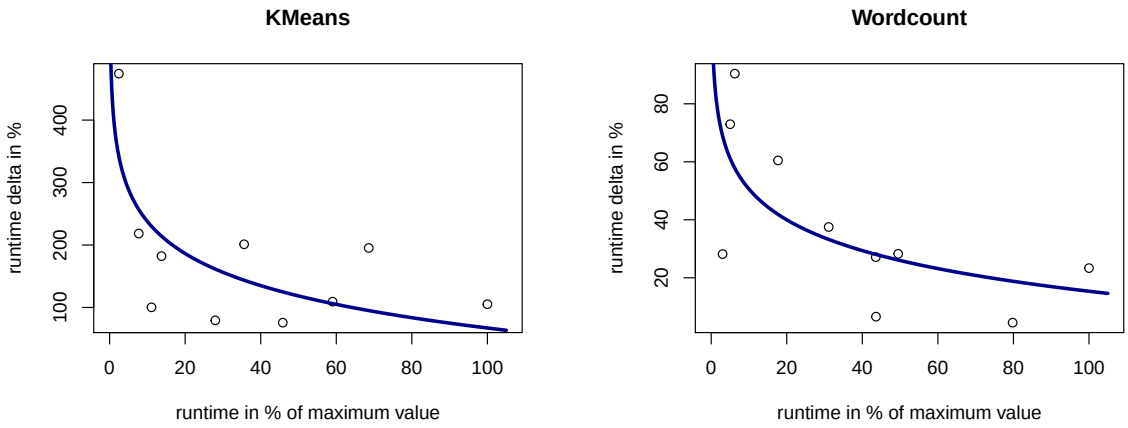


Figure 5.7: Runtime delta between prediction and measurement depending on input size

Figure 5.8: Flink dataflow graph of a Wordcount execution

lower range. Its 38% percent averaged delta runtime is also different among the input sizes and corresponding runtime values. They show no constant distribution, but also a reduction with an increasing runtime.

Instead of K-Means, Wordcount has a much simpler dataflow graph of the Flink execution as shown in figure 5.8, including only one map and one reduce phase, counting the words and combining the results. Although the complexity of these two applications is different they show a similar behavior in the deviation of the predictions and actual measurements.

The final conclusion to analyze this behavior is not possible within the scope of this thesis, but will be an important aspect of any future work. In which way the execution environment influences or even produces this behavior might be investigated by comparing these results to measurements in an exclusively used environment and also with CPU, network and disk load measurements on the virtual machines.

Environmental influence on job executions

The actual influence of the execution environment can not only be assumed in the overall deviation of the measurements, but also precisely analyzed in the series of measurements of the same configuration. One example of the measured differences is shown in figure 5.9, where three temporally consecutive measurement results of the application Wordcount, with 4 VMs each with 4 CPUs, are shown.

The overall runtime of the job execution on the master VM is displayed with the red bar. It is strongly dependent on the highest runtime of any of the 16 task managers, colored by the hosting VM. Because the input file was split into 16 parts, in each run two task

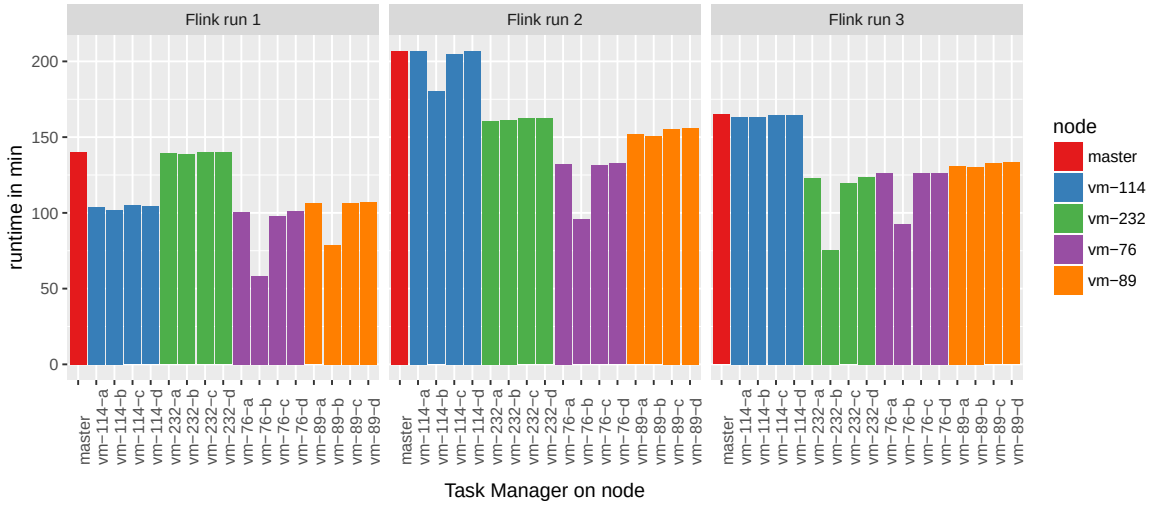


Figure 5.9: Wordcount runtime of task managers involved with huge offset

managers had a reduced input amount and hence a reduced runtime. All other tasks on the same vm had approximately the same runtime within one execution. However, as it can be seen, for example the blue *vm-114* has a difference in runtime between execution 1 and 2 of over 100 minutes and also over 100%. This VM might be influenced by external load on the underlying hardware node and therefore differs that much in the measured runtime value.

The assumption that such load on the environment influences all taken measurements might be an explanation for the high calculated standard deviation and uncertainty of the prediction presented in the previous chapter. Additional measurements of the hardware nodes could clarify this proposition and are recommended as a task of the future work to enhance any regression model based on the developed methodology.

This chapter presented runtime predictions of the instantiated applications and compared them to actual measured ones. Based on this comparison, both determined regression models could be validated. These positive findings will be used in the next chapter to evaluate the methodology.

6 Evaluation and Conclusion

This chapter will sum up the findings of this thesis, evaluate them and provide an outlook especially including future work proposals.

6.1 Methodology Evaluation

This thesis introduced a methodology to predict the runtime behavior of large scale data applications. It was exemplarily instantiated in a shared cloud environment with the applications K-Means and Wordcount, showing all details of the methodology. Beside the major tasks, all necessary statistical and mathematical principles of the function analysis and the used Multiple Linear Regression were explained and defined in corresponding equations. Besides the actual focused runtime prediction, this thesis could additionally introduce the prediction of an optimal configuration on a subset of the problem space.

Although both resulting regression models were obtained on high deviated measurements, they could be validated. The thereby predicted and measured runtime values could be located in an allowed distance defined by the model. These detected runtime differences between both values were valid to the statistical tolerance, but not satisfying for any user. This is not the fault of the used regression or even the methodology, it is only caused by the deviated measurements.

How good any prediction is at the end will always depend on the reliability of the underlying data. Furthermore, due to different hardware configurations of the used cloud nodes, an influencing load on each single node during the measurements and also a fluctuating network bandwidth, the initial measurements were error loaded. This could also be observed in the validation measurements and the seen prediction deviation from the actual measured runtime.

During the thesis all effort was invested to compensate these deviations. Measurements were taken recurrently, on other weekdays and at other hours. With the granularity of the defined problem space dimensions, the amount of required measurements was finally raised to a value where not all measurements could be taken with the needed repetitions or even not at all. This lead to several unmeasured configurations especially with the largest input files, which were last added to the configuration, whereas most other measurements with a lower size could to be covered at least once.

The fact that this could possibly lead to a weak spot was identified the first time when instantiating the methodology. Almost all tries to measure the missing runtime values failed, because the Flink job execution failed due to several *lost* task managers. What this error message meant in each single case could not be investigated, only that the communication between the master and the worker node was interrupted and therefore the complete job execution.

Nevertheless, the twice instantiated methodology resulted in regression models, which could be validated under consideration of the statistical circumstances. For this reason,

the developed methodology can also be successfully evaluated on the basis of these two applications. There exists no doubt that the methodology will also be reliable on other instantiations with even more complex implementations.

All described specifications are selected as generic as possible, to be suitable and applicable to all kinds of applications. The defined problem space covers all relevant alterable features for any implementation in a cloud infrastructure as shown.

Data streaming could not be included into the instantiations of the methodology so far. The possible shape of data streams will extend the presented configuration features. Data blocks might arrive equally distributed over time, in repetitive peaks or completely inconsistent. However, the developed methodology will be also applicable on these new characteristics, because the problem space is defined to be extendable with every cloud or application feature. Therefore, the thesis can be concluded with a very high confidence regarding following instantiations of the developed methodology.

6.2 Outlook and Future Work

The indented aim of the thesis was fulfilled with the developed and validated methodology.

To improve the resulting predictions and tighten the confidence and prediction interval, measurements taken in an unshared cloud environment should essentially scale down the uncertainty of the prediction. It is expected to improve the $R^2_{adj.}$ value of each regression model and minimize the delta runtime values.

Any extension on more applications should continually validate the methodology. An increasing complexity and hence more different map and reduce phases could also enhance the exploration of the general runtime behavior of any generic application in such environments. Also, the study of any communication patterns inside the executions as well as the different limitations of the phases will contribute to this aim. Measurements on the virtual machines regarding CPU, memory, disk or network load will open new aspects to probably find a general model.

The prediction and selection of an optimal cloud configuration can for example be enhanced by introducing weights on each dimension. This way the configuration can be selected with a prior view on specific dimensions as for example the amount of VMs. Also, a parameter could be added to predict the arising costs of the resources, especially for a commercial cloud environment.

With the decision to use Apache Flink within this thesis, the long-term aim was to include data-streaming applications. Both presented applications K-Means and Wordcount can also be executed with a streaming input. This might extend and refine the problem space with new dimensions representing this continual input and again increase the granularity of the methodology.

A appendix

A.1 Measured Runtime Values

The measured runtime values in milliseconds of each configuration are listed in the following tables, dependent on the input size.

A.1.1 K-Means

The K-Means input sizes were 50, 200, 500, 2,000 and 5,000 million points:

50 million points

VMs	CPUs	RAM	runtime	VMs	CPUs	RAM	runtime
1	1	4	1681516	1	4	4	1141947
1	1	4	1753094	1	4	4	975575
1	1	4	1737465	1	4	4	912992
1	1	4	1768235	1	4	4	902234
1	1	4	1674560	2	4	4	753595
1	1	4	1724480	2	4	4	621329
1	1	4	1730614	2	4	4	761454
1	1	4	2115085	2	4	4	542652
1	1	4	2121326	2	4	4	569292
1	1	4	2107748	2	4	4	623525
1	1	4	2122405	2	4	4	758765
1	1	4	2015098	2	4	4	648569
1	1	4	2033798	4	4	4	299741
1	1	4	2015587	4	4	4	337362
1	1	4	1994495	4	4	4	352141
2	1	4	1371401	4	4	4	388228
2	1	4	1452957	4	4	4	377016
2	1	4	1483307	4	4	4	389027
2	1	4	1320937	4	4	4	443424
2	1	4	1410581	4	4	4	319768
2	1	4	1379877	8	4	4	142387
2	1	4	1442312	8	4	4	153852
2	1	4	1427158	8	4	4	185458
2	1	4	1507325	8	4	4	144694
2	1	4	1443638	8	4	4	170406
2	1	4	1211462	8	4	4	154696
2	1	4	1249521	8	4	4	209130
4	1	4	704550	8	4	4	219645

VMs	CPUs	RAM	runtime	VMs	CPUs	RAM	runtime
4	1	4	732826	8	4	4	178328
4	1	4	694825	8	4	4	192420
4	1	4	740818	8	4	4	164532
4	1	4	770720	8	4	4	269116
4	1	4	792199	8	4	4	210109
4	1	4	820441	8	4	4	293631
4	1	4	783067	8	4	4	177842
4	1	4	745098	8	4	4	224254
4	1	4	817727	8	4	4	265361
4	1	4	668097	1	8	4	775460
4	1	4	659056	1	8	4	801222
4	1	4	724274	1	8	4	749034
4	1	4	731764	1	8	4	885082
8	1	4	392394	1	8	4	744484
8	1	4	375044	1	8	4	710062
8	1	4	376146	1	8	4	835738
8	1	4	363483	1	8	4	788816
8	1	4	357912	2	8	4	323316
8	1	4	384745	2	8	4	389198
8	1	4	349719	2	8	4	413808
8	1	4	379601	2	8	4	331354
8	1	4	407727	2	8	4	381053
8	1	4	349487	2	8	4	414484
8	1	4	439126	2	8	4	434228
8	1	4	349682	2	8	4	497274
8	1	4	370750	4	8	4	250608
8	1	4	344025	4	8	4	242974
8	1	4	348550	4	8	4	218957
1	2	4	1547373	4	8	4	247240
1	2	4	1419273	4	8	4	274556
1	2	4	1228719	4	8	4	275529
1	2	4	1125511	4	8	4	496344
1	2	4	1401887	4	8	4	304518
1	2	4	1328357	4	8	4	332001
1	2	4	1244015	4	8	4	346775
1	2	4	1137889	4	8	4	331788
2	2	4	704487	8	8	4	215282
2	2	4	689405	8	8	4	166558
2	2	4	697761	8	8	4	188318
2	2	4	761839	8	8	4	131179
2	2	4	767569	8	8	4	174840
2	2	4	742950	8	8	4	208057
2	2	4	707660	8	8	4	139591
2	2	4	739536	8	8	4	139552
4	2	4	470466	8	8	4	254494
4	2	4	506881	1	1	8	1731003

VMs	CPUs	RAM	runtime	VMs	CPUs	RAM	runtime
4	2	4	491510	1	1	8	1603449
4	2	4	619464	1	1	8	1657378
4	2	4	493179	1	2	8	1336683
4	2	4	451458	1	2	8	1214948
4	2	4	394970	1	2	8	1162573
4	2	4	773293	2	2	8	574782
8	2	4	352668	2	2	8	588399
8	2	4	222650	2	2	8	596639
8	2	4	328161	2	4	8	381184
8	2	4	271391	2	4	8	397981
8	2	4	314855	2	4	8	461587
8	2	4	294233	2	4	8	467110
8	2	4	268771	4	8	8	151931
8	2	4	283103	4	8	8	146101
1	4	4	906978	4	8	8	146067
1	4	4	801552	4	8	8	170636
1	4	4	933983	4	8	8	135953

Table A.1: measured runtime values in milliseconds of the use-case K-Means for an input size of 50 million points

200 million points

VMs	CPUs	RAM	runtime	VMs	CPUs	RAM	runtime
1	1	4	8967766	2	4	4	2322468
1	1	4	8933189	2	4	4	2040928
1	1	4	9461307	4	4	4	1631879
1	1	4	9196143	4	4	4	1708403
2	1	4	4990454	4	4	4	1793096
2	1	4	4406734	4	4	4	2013413
2	1	4	4811394	8	4	4	856067
2	1	4	4530012	8	4	4	1115133
4	1	4	3020876	8	4	4	1199191
4	1	4	2998829	8	4	4	1476587
4	1	4	2914393	8	4	4	962781
4	1	4	2847225	4	1	8	3030248
8	1	4	1681406	4	1	8	2622053
8	1	4	1646454	4	1	8	2858558
8	1	4	1867536	8	1	8	1739992
8	1	4	1754292	8	1	8	1687213
1	2	4	4641036	8	1	8	1664925
1	2	4	4506856	16	1	8	956268
1	2	4	4385864	16	1	8	1147737
1	2	4	4559397	16	1	8	1168452
1	4	4	4334644	4	2	8	1462419
1	4	4	4319130	4	2	8	1430367

VMs	CPUs	RAM	runtime	VMs	CPUs	RAM	runtime
1	4	4	4766410	4	2	8	1493186
1	4	4	4677943	8	2	8	849824
2	4	4	2574265	8	2	8	862638
2	4	4	2469414	8	2	8	888761

Table A.2: measured runtime values in milliseconds of the use-case K-Means for an input size of 200 million points

500 million points

VMs	CPUs	RAM	runtime	VMs	CPUs	RAM	runtime
1	1	4	22292650	2	4	4	5912260
1	1	4	21795971	2	4	4	6144979
1	1	4	21729832	2	4	4	7368523
1	1	4	21427771	2	4	4	7351840
1	1	4	19646850	2	4	4	5969472
1	1	4	22906253	2	4	4	7284816
1	1	4	21313680	2	4	4	5805154
1	1	4	21045502	4	4	4	4226358
2	1	4	13270681	4	4	4	3873059
2	1	4	13057327	4	4	4	3581347
2	1	4	11531655	4	4	4	3287163
2	1	4	13678751	4	4	4	3411184
2	1	4	11485286	4	4	4	3129130
2	1	4	12602346	4	4	4	3212769
2	1	4	11817405	8	4	4	2266003
2	1	4	11905793	8	4	4	2086800
4	1	4	7384081	8	4	4	2077102
4	1	4	7284891	8	4	4	2375380
4	1	4	7103078	8	4	4	2334141
4	1	4	6456781	8	4	4	2752013
4	1	4	6449561	8	4	4	2314284
4	1	4	6468292	1	8	4	9331698
4	1	4	6342517	1	8	4	9477089
4	1	4	6771926	1	8	4	9615628
8	1	4	3860829	1	8	4	9287161
8	1	4	3666761	1	8	4	9343793
8	1	4	3654587	1	8	4	10780125
8	1	4	3701354	1	8	4	11050710
8	1	4	4130525	1	8	4	10039399
8	1	4	3901856	2	8	4	4964663
8	1	4	3874996	2	8	4	4510911
8	1	4	3740767	2	8	4	4718002
16	1	4	2759629	2	8	4	4620842
16	1	4	2539313	2	8	4	4954238
16	1	4	2616921	2	8	4	5272468

VMs	CPUs	RAM	runtime	VMs	CPUs	RAM	runtime
1	2	4	14445197	2	8	4	6927583
1	2	4	14802033	2	8	4	5726300
1	2	4	13920646	4	8	4	3387056
1	2	4	14233633	4	8	4	3369487
1	2	4	14042151	4	8	4	3129871
1	2	4	14260088	4	8	4	3392412
1	2	4	14400820	4	8	4	3520574
1	2	4	14490294	4	8	4	4254310
2	2	4	8306870	4	8	4	3931163
2	2	4	8421878	8	8	4	2207551
2	2	4	8501599	8	8	4	2978653
2	2	4	8329325	8	8	4	2424826
2	2	4	9218429	8	8	4	2206641
2	2	4	7770299	8	8	4	2208634
2	2	4	7307607	8	8	4	2195253
2	2	4	7831820	8	8	4	2738368
4	2	4	5244562	8	8	4	2387646
4	2	4	4665760	8	8	4	2154244
4	2	4	4696269	1	1	8	20183408
4	2	4	4593162	1	1	8	24706580
4	2	4	4521025	1	1	8	22500917
4	2	4	4611485	1	1	8	25063869
4	2	4	4684256	2	1	8	12160791
4	2	4	4119967	2	1	8	11995009
8	2	4	2808439	2	1	8	12496007
8	2	4	2615311	2	1	8	12902593
8	2	4	2586110	2	1	8	9960043
8	2	4	2823704	2	1	8	10166042
8	2	4	2394742	2	1	8	9999795
8	2	4	3099013	4	1	8	7262198
8	2	4	2616683	4	1	8	7353341
8	2	4	2751637	4	1	8	6694321
1	4	4	9791927	4	1	8	6057379
1	4	4	9442999	4	1	8	7128640
1	4	4	9082389	4	1	8	6864639
1	4	4	9762517	4	1	8	6931793
1	4	4	12459720	8	1	8	3974457
1	4	4	12091157	8	1	8	4401595
1	4	4	10201490	8	1	8	4559721
1	4	4	9247666	8	1	8	4013323
2	4	4	6265576				

Table A.3: measured runtime values in milliseconds of the use-case K-Means for an input size of 500 million points

2,000 million points

VMs	CPUs	RAM	runtime	VMs	CPUs	RAM	runtime
1	1	8	88538035	8	2	8	10551399
1	1	8	79616355	8	2	8	11592442
2	1	8	52645154	1	4	8	47088865
2	1	8	46500016	1	4	8	48745152
2	1	8	52436541	1	4	8	43768286
4	1	8	30258392	2	4	8	33791022
4	1	8	30932978	2	4	8	30728890
8	1	8	14646341	2	4	8	27324173
8	1	8	16126120	4	4	8	16579567
8	1	8	16583429	4	4	8	15510054
16	1	8	10381263	4	4	8	14281609
16	1	8	10327493	8	4	8	11700689
16	1	8	11376681	16	4	8	6438838
1	2	8	55805878	1	1	16	90223995
1	2	8	64194252	2	1	16	40696911
1	2	8	66805381	8	1	16	15204575
4	2	8	15854887	4	2	16	17333536
4	2	8	15954049	4	2	16	16663838
4	2	8	16933199	8	2	16	11463327
8	2	8	10540952	8	2	16	8521653

Table A.4: measured runtime values in milliseconds of the use-case K-Means for an input size of 2,000 million points

5,000 million points

VMs	CPUs	RAM	runtime	VMs	CPUs	RAM	runtime
1	1	8	87829379	4	2	8	41658976
1	1	8	76158936	4	2	8	41237702
1	1	8	76648592	4	2	8	42123152
4	1	8	76344133	8	2	8	26868044
4	1	8	69028363	8	2	8	26774349
8	1	8	43684560	2	4	8	61814150
8	1	8	40885663	4	4	8	31747600
16	1	8	23592087	4	4	8	29308169
32	1	8	16471995	4	4	8	30310338
32	1	8	16644173	8	4	8	22330338
2	2	8	95033249	8	4	8	28715049
2	2	8	91698439				

Table A.5: measured runtime values in milliseconds of the use-case K-Means for an input size of 5,000 million points

A.1.2 Wordcount

The Wordcount input sizes were 176,935,868, 702,062,928, 1,645,401,881 and 14,934,742,463 words:

176,935,868 words

VMs	CPUs	RAM	runtime	VMs	CPUs	RAM	runtime
1	1	4	1726594	8	1	4	313112
1	1	4	1720914	8	1	4	298720
1	1	4	1742032	8	1	4	324981
1	1	4	1853828	8	1	4	274394
1	1	4	1824995	8	1	4	286878
1	1	4	1859408	8	1	4	341828
1	1	4	1845990	8	1	4	299292
1	1	4	1499119	8	1	4	290028
1	1	4	1581152	8	1	4	322460
1	1	4	1493300	1	1	8	1987504
1	1	4	1548692	1	1	8	1943962
1	1	4	1592227	1	1	8	1842878
2	1	4	1105764	2	1	8	1232587
2	1	4	1055523	2	1	8	1179705
2	1	4	1305348	2	1	8	1173190
2	1	4	1019021	1	2	8	3547973
2	1	4	798604	1	2	8	3462086
2	1	4	948286	1	2	8	3713849
2	1	4	949184	2	2	8	2321808
2	1	4	790670	2	2	8	2162374
2	1	4	1093650	2	2	8	1446863
2	1	4	1041921	4	2	8	987591
2	1	4	1068975	4	2	8	1157996
2	1	4	1060901	4	2	8	933583
4	1	4	628041	8	2	8	506567
4	1	4	662344	8	2	8	545896
4	1	4	625643	8	2	8	581105
4	1	4	554986	1	4	8	3795971
4	1	4	587893	1	4	8	3688272
4	1	4	500088	1	4	8	3781671
4	1	4	516815	2	4	8	1664605
4	1	4	479284	2	4	8	2229609
4	1	4	585625	2	4	8	2221643
4	1	4	577527	4	4	8	985550
4	1	4	539285	4	4	8	1240986
4	1	4	536080	4	4	8	1206946
8	1	4	334746	8	4	8	669695
8	1	4	338524	8	4	8	631978
8	1	4	347002	8	4	8	594273

VMs	CPUs	RAM	runtime	VMs	CPUs	RAM	runtime

Table A.6: measured runtime values in milliseconds of the use-case Wordcount for an input size of 176.9 million words

702,062,928 words

VMs	CPUs	RAM	runtime	VMs	CPUs	RAM	runtime
1	1	4	7088587	2	4	4	6432158
1	1	4	6766641	2	4	4	6478609
1	1	4	6892678	4	4	4	3486007
1	1	4	6871840	4	4	4	3569978
1	1	4	7395073	4	4	4	3947242
1	1	4	8211067	4	4	4	3347369
1	1	4	7183433	4	4	4	3514623
1	1	4	7059997	4	4	4	3858004
1	1	4	9271923	4	4	4	3349311
1	1	4	9928123	4	4	4	3312588
1	1	4	9379208	8	4	4	2211276
1	1	4	9642022	8	4	4	1713351
2	1	4	5402298	8	4	4	2125466
2	1	4	5485679	8	4	4	2074091
2	1	4	5481212	8	4	4	1784546
2	1	4	5876778	8	4	4	2351731
2	1	4	4460001	8	4	4	2381838
2	1	4	4723873	8	4	4	2305332
2	1	4	4482744	1	8	4	10568641
2	1	4	4911585	1	8	4	10112561
2	1	4	6253151	1	8	4	10652971
2	1	4	5210911	1	8	4	10425353
2	1	4	4698827	1	8	4	10103618
2	1	4	5318465	1	8	4	9709991
4	1	4	2491740	1	8	4	9645803
4	1	4	1897747	1	8	4	9510782
4	1	4	1790605	2	8	4	7353761
4	1	4	2010079	2	8	4	6967728
4	1	4	1889823	2	8	4	7594911
4	1	4	2559561	2	8	4	7248379
4	1	4	2166788	2	8	4	11278595
4	1	4	2170494	2	8	4	10292139
4	1	4	2968413	2	8	4	6865429
4	1	4	3018438	2	8	4	10984683
4	1	4	3113592	4	8	4	4938212
4	1	4	3055397	4	8	4	4663836
8	1	4	1399086	4	8	4	4840918
8	1	4	1340872	4	8	4	4574144
8	1	4	1283666	8	8	4	2568257

VMs	CPUs	RAM	runtime	VMs	CPUs	RAM	runtime
8	1	4	1320253	8	8	4	2557358
8	1	4	1271388	8	8	4	2393656
8	1	4	1166334	8	8	4	2472665
8	1	4	1312549	1	1	8	7547746
8	1	4	1383625	1	1	8	7544396
8	1	4	1866938	1	1	8	7782428
8	1	4	1372109	2	1	8	3738052
8	1	4	1386642	2	1	8	4131073
8	1	4	1693577	2	1	8	4001456
1	2	4	9996888	4	1	8	2545967
1	2	4	9870873	4	1	8	2306846
1	2	4	9539139	4	1	8	2120197
1	2	4	9518508	8	1	8	1057952
1	2	4	9137195	8	1	8	1433681
2	2	4	7701069	8	1	8	1274946
2	2	4	8305344	16	1	8	839723
2	2	4	8276044	16	1	8	1011150
2	2	4	7756900	16	1	8	992974
2	2	4	8097366	1	2	8	14183827
4	2	4	4262043	1	2	8	14600039
4	2	4	3777737	1	2	8	14558280
4	2	4	4382906	2	2	8	7449460
4	2	4	3884397	2	2	8	8793469
4	2	4	4275897	2	2	8	8600011
8	2	4	2424530	4	2	8	3687276
8	2	4	2355152	4	2	8	4420221
8	2	4	1975266	4	2	8	3568918
8	2	4	2364841	8	2	8	2247092
8	2	4	2059067	8	2	8	2272262
1	4	4	10184717	8	2	8	2253011
1	4	4	10368603	1	4	8	10440174
1	4	4	10225448	1	4	8	10222158
1	4	4	10370618	1	4	8	11087240
1	4	4	10276693	2	4	8	7213311
1	4	4	10372465	2	4	8	7093156
1	4	4	10896263	2	4	8	6995238
1	4	4	10616436	4	4	8	3847207
2	4	4	6316640	4	4	8	3591164
2	4	4	9148584	4	4	8	3281850
2	4	4	9499184	8	4	8	2400205
2	4	4	9787950	8	4	8	2527035
2	4	4	7056917	8	4	8	2278209
2	4	4	6420135				

Table A.7: measured runtime values in milliseconds of the use-case Wordcount for an input size of 702.1 million words

1,645,401,881 words

VMs	CPUs	RAM	runtime	VMs	CPUs	RAM	runtime
1	1	4	19140864	1	8	4	31331474
1	1	4	24268050	1	8	4	32897949
1	1	4	29549423	1	8	4	34835497
1	1	4	30551081	1	8	4	27501271
1	1	4	30101080	1	8	4	29010982
2	1	4	10885893	1	8	4	28693795
2	1	4	8695405	2	8	4	15521210
2	1	4	10833792	2	8	4	15880025
2	1	4	11865219	2	8	4	17163280
4	1	4	4532972	2	8	4	16713511
4	1	4	4555591	2	8	4	15998807
4	1	4	5791886	2	8	4	16319362
4	1	4	4729545	4	8	4	9387889
4	1	4	5612827	4	8	4	9089707
4	1	4	5530383	4	8	4	8296232
4	1	4	5757109	4	8	4	9193193
4	1	4	5770854	8	8	4	4290978
4	1	4	5707687	8	8	4	4719162
4	1	4	5538386	1	1	8	25154219
8	1	4	2565926	1	1	8	27489892
8	1	4	3021004	2	1	8	9349976
8	1	4	2455320	2	1	8	9450135
8	1	4	2915445	2	1	8	8768576
8	1	4	2732770	4	1	8	5071914
1	2	4	30532049	4	1	8	6513567
1	2	4	29051124	4	1	8	5237400
1	2	4	28045068	8	1	8	2490527
1	2	4	26883331	8	1	8	3241765
1	2	4	31376353	8	1	8	3592825
1	2	4	31296248	16	1	8	2230709
1	2	4	30156760	16	1	8	2208940
1	2	4	29954979	16	1	8	2456948
2	2	4	15916959	1	2	8	28817384
2	2	4	15471292	1	2	8	27117116
2	2	4	16117029	1	2	8	25140912
2	2	4	17098876	2	2	8	20221982
2	2	4	16550248	2	2	8	19318618
2	2	4	13787438	2	2	8	19981643
2	2	4	15325751	4	2	8	10222685
4	2	4	10643287	4	2	8	10272197
4	2	4	10214526	4	2	8	10436518
4	2	4	8003877	8	2	8	5398027
4	2	4	7877858	8	2	8	5688508
4	2	4	10397761	8	2	8	6066849

VMs	CPUs	RAM	runtime	VMs	CPUs	RAM	runtime
4	2	4	10466871	1	4	8	34348601
4	2	4	8041104	1	4	8	32519274
4	2	4	8114421	1	4	8	32475541
8	2	4	5055531	2	4	8	22280032
8	2	4	5016514	2	4	8	22911573
8	2	4	5278660	2	4	8	21858951
8	2	4	4967724	4	4	8	10636023
1	4	4	38669832	4	4	8	8432499
1	4	4	37768473	4	4	8	9467273
1	4	4	34895548	8	4	8	5803859
1	4	4	34379995	8	4	8	5747991
1	4	4	38599095	8	4	8	5861857
1	4	4	37963591	2	8	8	17565232
1	4	4	34346791	2	8	8	18827077
1	4	4	34173278	1	1	16	19526983
2	4	4	17899436	1	1	16	19285696
2	4	4	21802659	1	1	16	19907200
2	4	4	17992754	2	1	16	11094739
2	4	4	22672958	2	1	16	11435078
2	4	4	21277637	2	1	16	11410225
2	4	4	21397227	4	1	16	7010659
2	4	4	15342131	4	1	16	7217511
2	4	4	16448913	4	1	16	6808463
4	4	4	10695956	8	1	16	3689614
4	4	4	9816589	8	1	16	3847965
4	4	4	9724612	8	1	16	3201060
4	4	4	8404958	1	2	16	30687567
4	4	4	9897743	1	2	16	29331359
4	4	4	10536655	2	2	16	19272479
4	4	4	12425390	2	2	16	18570261
8	4	4	6301674	4	2	16	10435105
8	4	4	5461690	4	2	16	11332642
8	4	4	6128920	1	4	16	29293590

Table A.8: measured runtime values in milliseconds of the use-case Wordcount for an input size of 1,645.4 million words

14,934,742,463 words

VMs	CPUs	RAM	runtime	VMs	CPUs	RAM	runtime
4	1	8	72808449	16	1	8	20619916
8	1	8	36421062	16	1	8	18632285
8	1	8	39046755				

Table A.9: measured runtime values in milliseconds of the use-case Wordcount for an input size of 14,934.7 million words

A.2 Apache Flink Configuration

For all deployed Flink job and task managers, the following configuration was used. The values *RAM*, *CPUs* and the job manager's *IP address* were inserted by the used controlling framework depending on the actual measurement setup.

configuration	value
jobmanager.rpc.address:	*from OpenNebula*
jobmanager.rpc.port:	6123
jobmanager.heap.mb:	1024 * *RAM* * 0.8
jobmanager.web.port:	8081
blob.server.port:	7101-7150
taskmanager.rpc.port:	7161
taskmanager.data.port:	7162
taskmanager.heap.mb:	1024 * *RAM* * 0.8
taskmanager.numberOfTaskSlots:	*CPUs*
taskmanager.memory.preallocate:	false
akka.framesize:	300000000b
parallelism.default:	1

List of Figures

3.1 Graph of the methodology to gain the prediction parameter 10
3.2 High-dimensional problem space for runtime analysis 11

4.1 The Flink process model from [CKE+15] 15
4.2 Experiment cloud setup infrastructure 16
4.3 Example of K-Means iterations to find centroids 18
4.4 R-plot of the measurements in each dimension against runtime in minutes for the use case K-Means 20
4.5 Box-Whisker-Plot with the number of VMs plotted against the measured runtime in minutes for the use case K-Means with an input size of 500 million points 21
4.6 Standard deviation and mean of all measurements, use case K-Means 22
4.7 Regression line plot with overfitting and underfitting 25
4.8 Dimension-wise regression functions and transformations 28
4.9 Outlier and Leverage points 34
4.10 Cook's distance plot for observations in the use case K-Means 35
4.11 Regression function plots in R against *input* and *vms* 41
4.12 Regression function plots in R against $cpus \cdot vms$ and $ram \cdot vms$ 42
4.13 Correlation plot of regression dimension from R 45
4.14 Plot of the Confidence and Prediction Intervals depending on the number of *vms*, with set values for $cpus = 1$, $ram = 8$ and $input = 5e{+}09$ points 48

5.1 Validation process of instantiated Methodology by runtime comparison 53
5.2 Plot of predicted and measured runtimes as well as confidence and prediction intervals for the use case K-Means 56
5.3 Wordcount example of counting words 58
5.4 Cook's Distance plot for application Wordcount 60
5.5 Plot of predicted and measured runtimes as well as confidence and prediction intervals for the use case Wordcount 62
5.6 Excerpt of a Flink dataflow graph for a K-Means execution 64
5.7 Runtime delta between prediction and measurement depending on input size 66
5.8 Flink dataflow graph of a Wordcount execution 66
5.9 Wordcount runtime of task managers involved with huge offset 67

Bibliography

[Aka11] Akaike, Hirotugu: Akaike's information criterion. In: *International Encyclopedia of Statistical Science*. Springer, 2011, S. 25–25

[ALC+17] Alipourfard, Omid ; Liu, Hongqiang H. ; Chen, Jianshu ; Venkataraman, Shivaram ; Yu, Minlan ; Zhang, Ming: CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics. In: *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*, 2017, 469–482

[BHL+12] Bronstein, Ilja N. ; Hromkovic, Juraj ; Luderer, Bernd ; Schwarz, Hans-Rudolf ; Blath, Jochen ; Schied, Alexander ; Dempe, Stephan ; Wanka, Gert ; Gottwald, Siegfried: *Taschenbuch der mathematik*. Bd. 1. Springer-Verlag, 2012

[Caf15] Caffo, Brian: Regression Models for data science in R. In: *Leanpub, British Columbia, Canada* (2015)

[CKE+15] Carbone, Paris ; Katsifodimos, Asterios ; Ewen, Stephan ; Markl, Volker ; Haridi, Seif ; Tzoumas, Kostas: Apache flink: Stream and batch processing in a single engine. In: *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 36 (2015), Nr. 4

[Coo11] Cook, R. D.: Cook's Distance. In: *International Encyclopedia of Statistical Science*. Springer, 2011, S. 301–302

[DG08] Dean, Jeffrey ; Ghemawat, Sanjay: MapReduce: simplified data processing on large clusters. In: *Communications of the ACM* 51 (2008), Nr. 1, S. 107–113

[EB16] Escobar, R. ; Boppana, R. V.: Performance Prediction of Parallel Applications Based on Small-Scale Executions. In: *2016 IEEE 23rd International Conference on High Performance Computing (HiPC)*, 2016, S. 362–371

[HJ15] Harrell Jr, Frank E.: *Regression modeling strategies: with applications to linear models, logistic and ordinal regression, and survival analysis*. Springer, 2015

[LTX16] Li, T. ; Tang, J. ; Xu, J.: Performance Modeling and Predictive Scheduling for Distributed Stream Data Processing. In: *IEEE Transactions on Big Data* 2 (2016), Dec, Nr. 4, S. 353–364. `http://dx.doi.org/10.1109/TBDATA.2016.2616148`. – DOI 10.1109/TBDATA.2016.2616148

[Pri17] Princeton University Library (Hrsg.): *Interpreting Regression Output*. Princeton University Library, 2017. `http://dss.princeton.edu/online_help/analysis/interpreting_regression.htm`

[Rei11] Reid, Nancy: Likelihood. In: *International Encyclopedia of Statistical Science*. Springer, 2011, S. 739–741

[She09] Sheather, Simon: *A modern Approach to Regression with R*. Springer Science & Business Media, 2009

[SW65] Shapiro, Samuel S. ; Wilk, Martin B.: An analysis of variance test for normality (complete samples). In: *Biometrika* 52 (1965), Nr. 3/4, S. 591–611

[VYF+16] Venkataraman, Shivaram ; Yang, Zongheng ; Franklin, Michael ; Recht, Benjamin ; Stoica, Ion: *Ernest: efficient performance prediction for large-scale advanced analytics*. Santa Clara, CA, 2016